

H A N D B O O K O F

AUTOMATED

VOLUME I

REASONING

ALAN ROBINSON AND ANDREI VORONKOV
EDITORS

NORTH * HOLLAND

HANDBOOK OF AUTOMATED
REASONING

VOLUME I

HANDBOOK OF AUTOMATED REASONING

VOLUME I

Editors

Alan Robinson

and

Andrei Voronkov



ELSEVIER

AMSTERDAM · LONDON · NEW YORK
OXFORD · PARIS · SHANNON · TOKYO



THE MIT PRESS
CAMBRIDGE, MASSACHUSETTS

ELSEVIER SCIENCE B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

Co-publishers for the United States and Canada:

The MIT Press
5 Cambridge Center, Cambridge, MA 02142, USA

Elsevier Science B.V.
ISBN: 0-444-82949-0 (Volume I)
ISBN: 0-444-50813-9 (Set of Vols. I + II)

The MIT Press
ISBN: 0-262-18221-1 (Volume I)
ISBN: 0-262-18223-8 (Set of Vols. I + II)

Library of Congress Control Number: 2001090839

© 2001 Elsevier Science B.V. All rights reserved.

This work is protected under copyright by Elsevier Science, and the following terms and conditions apply to its use:

Photocopying:

Single photocopies of single chapters may be made for personal use as allowed by national copyright laws. Permission of the Publisher and payment of a fee is required for all other photocopying, including multiple or systematic copying, copying for advertising or promotional purposes, resale, and all forms of document delivery. Special rates are available for educational institutions that wish to make photocopies for non-profit educational classroom use.

Permissions may be sought directly from Elsevier Science Global Rights Department, PO Box 800, Oxford OX5 1DX, UK; phone: (+44) 1865 843830, fax: (+44) 1865 853333, e-mail: permissions@elsevier.co.uk. You may also contact Global Rights directly through Elsevier's home page (<http://www.elsevier.nl>), by selecting 'Obtaining Permissions'.

In the USA, users may clear permissions and make payments through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA; phone: (+1) 978 7508400, fax: (+1) 978 7504744, and in the UK through the Copyright Licensing Agency Rapid Clearance Service (CLARCS), 90 Tottenham Court Road, London W1P 0LP, UK; phone: (+44) 207 631 5555; fax: (+44) 207 631 5500. Other countries may have a local reprographic rights agency for payments.

Derivative Works:

Tables of contents may be reproduced for internal circulation, but permission of Elsevier Science is required for external resale or distribution of such material. Permission of the Publisher is required for all other derivative works, including compilations and translations.

Electronic Storage or Usage:

Permission of the Publisher is required to store or use electronically any material contained in this work, including any chapter or part of a chapter.

Except as outlined above, no part of this work may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without prior written permission of the Publisher. Address permissions requests to: Elsevier Science Global Rights Department, at the mail, fax and e-mail addresses noted above.

Notice:

No responsibility is assumed by the Publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the material herein. Because of rapid advances in the medical sciences, in particular, independent verification of diagnoses and drug dosages should be made.

Ⓢ The paper used in this publication meets the requirements of ANSI/NISO Z39.48-1992 (Permanence of Paper).

Printed in The Netherlands

Contents

Volume 1

Part I History

CHAPTER 1. THE EARLY HISTORY OF AUTOMATED DEDUCTION	3
<i>Martin Davis</i>	
1 Presburger's Procedure	5
2 Newell, Shaw & Simon, and H. Gelernter	6
3 First-Order Logic	7
Bibliography	12
Index	15

Part II Classical Logic

CHAPTER 2. RESOLUTION THEOREM PROVING	19
<i>Leo Bachmair and Harald Ganzinger</i>	
1 Introduction	21
2 Preliminaries	22
3 Standard Resolution	28
4 A Framework for Saturation-Based Theorem Proving	34
5 General Resolution	46
6 Basic Resolution Strategies	59
7 Refined Techniques for Defining Orderings and Selection Functions . .	66
8 Global Theorem Proving Methods	84
9 First-Order Resolution Methods	89
10 Effective Saturation of First-Order Theories	91
11 Concluding Remarks	93
Bibliography	94
Index	98

CHAPTER 3. TABLEAUX AND RELATED METHODS	101
<i>Reiner Hähnle</i>	
1 Introduction	103
2 Preliminaries	104
3 The Tableau Method	107
4 Clause Tableaux	125
5 Tableaux as a Framework	152
6 Comparing Calculi	164
7 Historical Remarks & Resources	167
Bibliography	168
Notation	176
Index	177
CHAPTER 4. THE INVERSE METHOD	179
<i>Anatoli Degtyarev and Andrei Voronkov</i>	
1 Introduction	181
2 Preliminaries	185
3 Cooking classical logic	186
4 Applying the recipe to nonclassical logics	209
5 Naming and connections with resolution	219
6 Season your meal: strategies and redundancies	232
7 Path calculi	233
8 Logics without the contraction rules	255
9 Conclusion	260
Bibliography	264
Index	270
CHAPTER 5. NORMAL FORM TRANSFORMATIONS	273
<i>Matthias Baaz, Uwe Egly, and Alexander Leitsch</i>	
1 Introduction	275
2 Notation and Definitions	278
3 On the Concept of Normal Form	287
4 Equivalence-Preserving Normal Forms	289
5 Skolem Normal Form	295
6 Conjunctive Normal Form	306
7 Normal Forms in Nonclassical Logics	323
8 Conclusion	328
Bibliography	328
Index	332

CHAPTER 6. COMPUTING SMALL CLAUSE NORMAL FORMS	335
<i>Andreas Nonnengart and Christoph Weidenbach</i>	
1 Introduction	337
2 Preliminaries	338
3 Standard CNF-Translation	340
4 Formula Renaming	347
5 Skolemization	352
6 Simplification	359
7 Bibliographic Notes	363
8 Implementation Notes	364
Bibliography	365
Index	367

Part III Equality and other theories

CHAPTER 7. PARAMODULATION-BASED THEOREM PROVING	371
<i>Robert Nieuwenhuis and Albert Rubio</i>	
1 About this chapter	373
2 Preliminaries	380
3 Paramodulation calculi	385
4 Saturation procedures	399
5 Paramodulation with constrained clauses	414
6 Paramodulation with built-in equational theories	421
7 Symbolic constraint solving	425
8 Extensions	427
9 Perspectives	429
Bibliography	432
Index	440
CHAPTER 8. UNIFICATION THEORY	445
<i>Franz Baader and Wayne Snyder</i>	
1 Introduction	447
2 Syntactic unification	450
3 Equational unification	469
4 Syntactic methods for <i>E</i> -unification	488
5 Semantic approaches to <i>E</i> -unification	503
6 Combination of unification algorithms	513
7 Further topics	519
Bibliography	521
Index	531

CHAPTER 9. REWRITING	535
<i>Nachum Dershowitz and David A. Plaisted</i>	
1 Introduction	537
2 Terminology	541
3 Normal Forms and Validity	544
4 Termination Properties	546
5 Church-Rosser Properties	559
6 Completion	567
7 Relativized Rewriting	574
8 Equational Theorem Proving	581
9 Conditional Rewriting	585
10 Programming	593
Bibliography	597
Index	608
CHAPTER 10. EQUALITY REASONING IN SEQUENT-BASED CALCULI	611
<i>Anatoli Degtyarev and Andrei Voronkov</i>	
1 Introduction	613
2 Translation of logic with equality into logic without equality	628
3 Free variable systems	637
4 Early history	644
5 Simultaneous rigid E -unification	646
6 Incomplete procedures for rigid E -unification	653
7 Sequent-based calculi and paramodulation	660
8 Equality elimination	667
9 Equality reasoning in nonclassical logics	679
10 Conclusion and open problems	691
Bibliography	693
Calculi and inference rules	703
Index	704
CHAPTER 11. AUTOMATED REASONING IN GEOMETRY	707
<i>Shang-Ching Chou and Xiao-Shan Gao</i>	
1 A history review of automated reasoning in geometry	709
2 Algebraic approaches to automated reasoning in geometry	712
3 Coordinate-free approaches to automated reasoning in geometry	732
4 AI approaches to automated reasoning in geometry	734
5 Final remarks	740
Bibliography	741
Index	749

CHAPTER 12. SOLVING NUMERICAL CONSTRAINTS	751
<i>Alexander Bockmayr and Volker Weispfenning</i>	
1 Introduction	753
2 Linear constraints over fields	758
3 Linear diophantine constraints	779
4 Non-linear constraints over continuous domains	802
5 Non-linear diophantine constraints	819
Bibliography	823
Index	838

Part IV Induction

CHAPTER 13. THE AUTOMATION OF PROOF BY MATHEMATICAL INDUCTION	845
<i>Alan Bundy</i>	
1 Introduction	847
2 Induction Rules	848
3 Recursive Definitions and Datatypes	851
4 Inductive Proof Techniques	855
5 Theoretical Limitations of Inductive Inference	863
6 Special Search Control Problems	865
7 Rippling	876
8 The Productive Use of Failure	890
9 Existential Theorems	894
10 Interactive Theorem Proving	898
11 Inductive Theorem Provers	900
12 Conclusion	903
Bibliography	904
Main Index	909
Name Index	911

CHAPTER 14. INDUCTIONLESS INDUCTION	913
<i>Hubert Comon</i>	
1 Introduction	915
2 Formal background	919
3 General Setting of the Inductionless Induction Method	925
4 Inductive completion methods	927
5 Examples of Axiomatizations \mathcal{A} from the Literature	938
6 Ground Reducibility	948
7 A comparison between inductive proofs and proofs by consistency	957
Bibliography	958
Index	961

Concept Index	963
----------------------	------------

Volume 2

Part V Higher-order logic and logical frameworks

CHAPTER 15. CLASSICAL TYPE THEORY	965
<i>Peter B. Andrews</i>	
1 Introduction to type theory	967
2 Metatheoretical foundations	977
3 Proof search	987
4 Conclusion	998
Bibliography	999
Index	1005
CHAPTER 16. HIGHER-ORDER UNIFICATION AND MATCHING	1009
<i>Gilles Dowek</i>	
1 Type Theory and Other Set Theories	1011
2 Simply Typed λ -calculus	1018
3 Undecidability	1024
4 Huet's Algorithm	1028
5 Scopes Management	1035
6 Decidable Subcases	1041
7 Unification in λ -calculus with Dependent Types	1049
Bibliography	1054
Index	1061
CHAPTER 17. LOGICAL FRAMEWORKS	1063
<i>Frank Pfenning</i>	
1 Introduction	1065
2 Abstract syntax	1067
3 Judgments and deductions	1075
4 Meta-programming and proof search	1095
5 Representing meta-theory	1108
6 Appendix: the simply-typed λ -calculus	1119
7 Appendix: the dependently typed λ -calculus	1123
8 Conclusion	1130
Bibliography	1135
Index	1145
CHAPTER 18. PROOF-ASSISTANTS USING DEPENDENT TYPE SYSTEMS	1149
<i>Henk Barendregt and Herman Geuvers</i>	
1 Proof checking	1151
2 Type-theoretic notions for proof checking	1153
3 Type systems for proof checking	1180

4	Proof-development in type systems	1211
5	Proof assistants	1223
	Bibliography	1230
	Index	1235
	Name index	1238

Part VI Nonclassical logics

CHAPTER 19. NONMONOTONIC REASONING: TOWARDS EFFICIENT CALCULI AND IMPLEMENTATIONS	1241
--	------

Jürgen Dix, Ulrich Furbach, and Ilkka Niemelä

1	General Nonmonotonic Logics	1244
2	Automating General Nonmonotonic Logics	1260
3	From Automated Reasoning to Disjunctive Logic Programming	1280
4	Nonmonotonic Semantics of Logic Programs	1297
5	Implementing Nonmonotonic Semantics	1311
6	Benchmarks	1332
7	Conclusion	1340
	Bibliography	1341
	Index	1352

CHAPTER 20. AUTOMATED DEDUCTION FOR MANY-VALUED LOGICS	1355
--	------

Matthias Baaz, Christian G. Fermüller, and Gernot Salzer

1	Introduction	1357
2	What is a many-valued logic?	1358
3	Classification of proof systems for many-valued logics	1361
4	Signed logic: reasoning classically about finitely-valued logics	1368
5	Signed resolution	1377
6	An example	1389
7	Optimization of transformation rules	1393
8	Remarks on infinitely-valued logics	1395
	Bibliography	1396
	Index	1401

CHAPTER 21. ENCODING TWO-VALUED NONCLASSICAL LOGICS IN CLASSICAL LOGIC	1403
---	------

*Hans Jürgen Ohlbach, Andreas Nonnengart, Maarten de Rijke, and
Dov M. Gabbay*

1	Introduction	1405
2	Background	1410
3	Encoding consequence relations	1419
4	The standard relational translation	1423
5	The functional translation	1440

6	The semi-functional translation	1455
7	Variations and alternatives	1465
8	Conclusion	1475
	Bibliography	1477
	Index	1484
CHAPTER 22. CONNECTIONS IN NONCLASSICAL LOGICS		1487
<i>Arild Waaler</i>		
1	Introduction	1489
2	Prelude: Connections in classical first-order logic	1491
3	Labelled systems	1516
4	Propositional intuitionistic logic	1528
5	First-order intuitionistic logic	1545
6	Normal modal logics up to S4	1553
7	The S5 family	1567
	Bibliography	1573
	Index	1577
 Part VII Decidable classes and model building		
CHAPTER 23. REASONING IN EXPRESSIVE DESCRIPTION LOGICS		1581
<i>Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi</i>		
1	Introduction	1583
2	Description Logics	1586
3	Description Logics and Propositional Dynamic Logics	1593
4	Unrestricted Model Reasoning	1598
5	Finite Model Reasoning	1610
6	Beyond Basic Description Logics	1619
7	Conclusions	1626
	Bibliography	1626
	Index	1633
CHAPTER 24. MODEL CHECKING		1635
<i>Edmund M. Clarke and Bernd-Holger Schlingloff</i>		
1	Introduction	1637
2	Logical Languages, Expressiveness	1641
3	Second Order Languages	1654
4	Model Transformations and Properties	1670
5	Equivalence reductions	1681
6	Completeness	1689
7	Decision Procedures	1700
8	Basic Model Checking Algorithms	1711

9	Modelling of Reactive Systems	1724
10	Symbolic Model Checking	1735
11	Partial Order Techniques	1751
12	Bounded Model Checking	1755
13	Abstractions	1759
14	Compositionality and Modular Verification	1764
15	Further Topics	1767
	Bibliography	1774
	Index	1788

CHAPTER 25. RESOLUTION DECISION PROCEDURES 1791

*Christian G. Fermüller, Alexander Leitsch, Ulrich Hustadt, and
Tanel Tammet*

1	Introduction	1793
2	Notation and definitions	1794
3	Decision procedures based on ordering refinements	1802
4	Hyperresolution as decision procedure	1814
5	Resolution decision procedures for description logics	1830
6	Related work	1842
	Bibliography	1843
	Index	1847

Part VIII Implementation

CHAPTER 26. TERM INDEXING 1853

R. Sekar, I.V. Ramakrishnan, and Andrei Voronkov

1	Introduction	1855
2	Background	1859
3	Data structures for representing terms and indexes	1866
4	A common framework for indexing	1870
5	Path indexing	1875
6	Discrimination trees	1883
7	Adaptive automata	1891
8	Automata-driven indexing	1900
9	Code trees	1908
10	Substitution trees	1917
11	Context trees	1922
12	Unification factoring	1924
13	Multiterm indexing	1927
14	Issues in perfect filtering	1934
15	Indexing modulo AC-theories	1939
16	Elements of term indexing	1943
17	Indexing in practice	1951

18 Conclusion	1955
Bibliography	1957
Index	1962
CHAPTER 27. COMBINING SUPERPOSITION, SORTS AND SPLITTING	1965
<i>Christoph Weidenbach</i>	
1 What This Chapter is (not) About	1967
2 Foundations	1969
3 A First Simple Prover	1973
4 Inference and Reduction Rules	1981
5 Global Design Decisions	2000
Bibliography	2008
A Links to Saturation Based Provers	2011
Index	2012
CHAPTER 28. MODEL ELIMINATION AND CONNECTION	
TABLEAU PROCEDURES	2015
<i>Reinhold Letz and Gernot Stenz</i>	
1 Introduction	2017
2 Clausal Tableaux and Connectedness	2018
3 Further Structural Refinements of Clausal Tableaux	2036
4 Global Pruning Methods in Model Elimination	2040
5 Shortening of Proofs	2049
6 Completeness of Connection Tableaux	2062
7 Architectures of Model Elimination Implementations	2070
8 Implementation of Refinements by Constraints	2092
9 Experimental Results	2102
10 Outlook	2107
Bibliography	2109
Index	2113
Concept index	2115

Preface

Automated reasoning has matured into one of the most advanced areas of computer science. During the half-century since the pioneering publications of the 1950s, significant progress has been achieved both in its theory and practice, culminating in a completely automatic solution of the Robbins Problem by the theorem prover EQP implemented by Bill McCune. This problem in algebra had remained open for over 50 years despite repeated attempts of mathematicians to solve it.

Several theoretical results, ideas, and techniques contributed to the Robbins Problem solution. We mention only a few: equational unification, and in particular AC-unification (Chapter 8 of this Handbook), completion procedures and notions of redundancy (Chapters 2 and 7), the basic strategy (Chapter 7), and term indexing (Chapter 26).

This Handbook presents overviews of the fundamental notions, techniques, ideas, and methods developed and used in automated reasoning and its applications, which are used in many areas of computer science, including software and hardware verification, logic and functional programming, formal methods, knowledge representation, deductive databases, and artificial intelligence..

The idea of making this Handbook originated during a visit by the first editor to the Computing Science Department of Uppsala University in 1996, where the second editor was working at the time. The idea was then presented at the Dagstuhl workshop on Deduction in 1997, after which the work began. It has taken four years to put together all the papers in their current form. Over 2000 email messages were exchanged between the editors and the authors.

The material included in the Handbook is intended to cover most of the areas in automated deduction, from theory to implementation. Nearly every chapter can be used as a basis for an undergraduate or a postgraduate course. In fact, some of them have already been so used. The chapters contain both basic and advanced material. It was deliberately decided also to include material that bridges the gap between the traditional automated reasoning (as presented at the CADE conferences) and related areas. Examples are model checking (Chapter 24), nonmonotonic reasoning (Chapter 19), numerical constraints (Chapter 12), description logics (Chapter 23), and implementation of declarative programming languages (Chapter 26).

To help the reader navigate through a large amount of material in the Handbook, the global concept index is provided at the end. It contains references to the pages containing the main notions and concepts introduced in different chapters.

The structure of the book is as follows.

Part I consists of a single chapter: an overview of the *early history* of automated deduction by Martin Davis.

Part II presents reasoning methods in *first-order logic*. Two most popular methods: *resolution* and *semantic tableaux* are discussed in Chapters 2 by Leo Bachmair and Harald Ganzinger and 3 by Reiner Hähnle. Nearly all existing implementations of first-order theorem provers are based on variants of one of these methods. The *inverse method*, both for classical and nonclassical logics, is introduced in Chapter 4 by Anatoli Degtyarev and Andrei Voronkov. Systems implementing first-order logic usually transform the goal formula into a clausal normal form. Chapters 5 by Matthias Baaz, Uwe Egly, and Alexander Leitsch, and 6 by Andreas Nonnengart and Christoph Weidenbach, discuss *short normal forms*, both from the theoretical and practical viewpoints.

Part III is dedicated to *equality and other built-in theories*. The first four chapters of this part discuss reasoning with equality and related subjects: Chapter 7 by Robert Nieuwenhuis and Albert Rubio deals with *paramodulation-based reasoning*, Chapter 8 by Franz Baader and Wayne Snyder presents *unification theory*, Chapter 9 by Nachum Dershowitz and David A. Plaisted overviews *rewriting*, and Chapter 10 by Anatoli Degtyarev and Andrei Voronkov discusses *equality reasoning in tableau-based and sequent-based calculi*. The next two chapters treat other important theories: Chapter 11 by Shang-Ching Chou and Xiao-Shan Gao presents *theorem proving in geometry*, while Chapter 12 by Alexander Bockmayr and Volker Weispfenning overviews methods of *solving numerical constraints*.

Part IV discusses methods of automated reasoning using *induction*. Chapter 13 by Alan Bundy gives a general introduction to induction, then Chapter 14 by Hubert Comon presents the so-called “*inductionless induction*” where induction is implemented using equational reasoning.

Part V discusses higher-order logic, which is used in a number of automatic and interactive proof-development systems. This part begins with two fundamental Chapters 15 by Peter Andrews and 16 by Gilles Dowek introducing, respectively, *classical type theory* and *higher-order unification*. The next two Chapters 17 by Frank Pfenning, and 18 by Henk Barendregt and Herman Geuvers discuss variants of higher-order logic used in two kinds of interactive systems: *logical frameworks* and *proof-assistants* using dependent type systems.

Part VI presents automated reasoning in *nonclassical logics*. Chapter 19 by Jürgen Dix, Ulrich Furbach, and Ilkka Niemelä is devoted to *nonmonotonic reasoning*, while Chapter 20 by Matthias Baaz, Christian G. Fermüller, and Gernot Salzer to reasoning in *many-valued logics*. The next two Chapters 21 by Hans Jürgen Ohlbach, Andreas Nonnengart, Maarten de Rijke, and Dov M. Gabbay, and 22 by Arild Waaler discuss reasoning methods for a wide range of logics whose semantics is characterized by the possible worlds semantics, for example, intuitionistic and modal logics. They discuss *translation into first-order classical logic* and the *connection method*,

respectively. Also highly relevant to reasoning in nonclassical logics are Chapters 4 and 23 put in other Parts of this Handbook.

Part VII deals with *decidable classes and model building*. The first two chapters concern two areas that have only recently emerged and which are now used in a number of applications: Chapter 24 by Edmund M. Clarke and Bernd-Holger Schlinghoff gives an overview of *model checking*, Chapter 23 by Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi discusses reasoning in expressive *description logics*. In Chapter 25 Christian G. Fermüller, Alexander Leitsch, Ullrich Hustadt, and Tanel Tammet present *resolution-based decision procedures* for various classes of first-order formulas.

Part VIII deals with *implementation-related* questions. In Chapter 26 R. Sekar, I.V. Ramakrishnan, and Andrei Voronkov give an overview of *term indexing* used in implementing not only first-order theorem provers but also logic and functional programming languages. The next two Chapters 27 by Christoph Weidenbach and 28 by Reinhold Letz and Gernot Stenz discuss implementation of, respectively, *resolution-based* and *model elimination-based theorem provers*.

The Web page

<http://www.cs.man.ac.uk/~voronkov/handbook-ar/index.html>

contains further material related to this book. All comments and corrections should be sent to the second editor by email voronkov@cs.man.ac.uk or v@ronkov.com.

Alan Robinson, Andrei Voronkov
Greenfield and Manchester, 14 February 2001.

List of contributors

This list contains the contributors of this Handbook with the corresponding chapter numbers.

Peter B. Andrews	15
<i>(Carnegie Mellon University, USA)</i>	
Franz Baader	8
<i>(RWTH Aachen, Germany)</i>	
Matthias Baaz	5, 20
<i>(Technische Universität Wien, Austria)</i>	
Leo Bachmair	2
<i>(State University of New York at Stony Brook, USA)</i>	
Henk Barendregt	18
<i>(University of Nijmegen, the Netherlands)</i>	
Alexander Bockmayr	12
<i>(LORIA, France)</i>	
Alan Bundy	13
<i>(University of Edinburgh, UK)</i>	
Diego Calvanese	23
<i>(Università di Roma "La Sapienza", Italy)</i>	
Shang-Ching Chou	11
<i>(Wichita State University, USA)</i>	
Edmund M. Clarke	24
<i>(Carnegie Mellon University, USA)</i>	
Hubert Comon	14
<i>(Ecole Normale Supérieure de Cachan, France)</i>	
Martin Davis	1
<i>(Courant Institute of Mathematical Sciences, USA)</i>	
Giuseppe De Giacomo	23
<i>(Università di Roma "La Sapienza", Italy)</i>	
Anatoli Degtyarev	4, 10
<i>(University of Liverpool, UK)</i>	
Nachum Dershowitz	9
<i>(Tel Aviv University, Israel)</i>	
Jürgen Dix	19
<i>(University of Manchester, UK)</i>	

Gilles Dowek	16
<i>(INRIA Rocquencourt, France)</i>	
Uwe Egly	5
<i>(Technische Universität Wien, Austria)</i>	
Christian G. Fermüller	20, 25
<i>(Technische Universität Wien, Austria)</i>	
Ulrich Furbach	19
<i>(Universität Koblenz-Landau, Germany)</i>	
Dov M. Gabbay	21
<i>(King's College London, UK)</i>	
Harald Ganzinger	2
<i>(Max-Planck-Institut für Informatik, Germany)</i>	
Xiao-Shan Gao	11
<i>(Institute of Systems Science, Academia Sinica, China)</i>	
Herman Geuvers	18
<i>(University of Nijmegen, the Netherlands)</i>	
Reiner Hähnle	3
<i>(Chalmers University, Sweden)</i>	
Ullrich Hustadt	25
<i>(University of Liverpool, UK)</i>	
Alexander Leitsch	5, 25
<i>(Technische Universität Wien, Austria)</i>	
Maurizio Lenzerini	23
<i>(Università di Roma "La Sapienza", Italy)</i>	
Reinhold Letz	28
<i>(Technische Universität München, Germany)</i>	
Daniele Nardi	23
<i>(Università di Roma "La Sapienza", Italy)</i>	
Ilkka Niemelä	19
<i>(Helsinki University of Technology, Finland)</i>	
Robert Nieuwenhuis	7
<i>(Technical University of Catalonia, Spain)</i>	
Andreas Nonnengart	6, 21
<i>(DFKI, Germany)</i>	
Hans Jürgen Ohlbach	21
<i>(Ludwig-Maximilians-Universität München, Germany)</i>	
Frank Pfenning	17
<i>(Carnegie Mellon University, USA)</i>	
David A. Plaisted	9
<i>(University of North Carolina at Chapel Hill, USA)</i>	

I.V. Ramakrishnan	26
<i>(State University of New York at Stony Brook, USA)</i>	
Maarten de Rijke	21
<i>(University of Amsterdam, the Netherlands)</i>	
Alan Robinson	0
<i>(USA)</i>	
Albert Rubio	7
<i>(Technical University of Catalonia, Spain)</i>	
Gernot Salzer	20
<i>(Technische Universität Wien, Austria)</i>	
Bernd-Holger Schlingloff	24
<i>(Universität Bremen, Germany)</i>	
R. Sekar	26
<i>(State University of New York at Stony Brook, USA)</i>	
Wayne Snyder	8
<i>(Boston University, USA)</i>	
Gernot Stenz	28
<i>(Technische Universität München, Germany)</i>	
Tanel Tammet	25
<i>(Tallinn University of Technology, Estonia)</i>	
Andrei Voronkov	0, 4, 10, 26
<i>(University of Manchester, UK)</i>	
Arild Waaler	22
<i>(University of Oslo, Norway)</i>	
Christoph Weidenbach	6, 27
<i>(Max-Planck-Institut für Informatik and Opel, Germany)</i>	
Volker Weispfenning	12
<i>(Universität Passau, Germany)</i>	

List of second readers

This list contains the second readers of this Handbook with the corresponding chapter numbers.

Peter B. Andrews	1
<i>(Carnegie Mellon University, USA)</i>	
Peter Baumgartner	28
<i>(Universität Koblenz-Landau, Germany)</i>	
Wolfgang Bibel	1
<i>(Technische Universität Darmstadt, Germany)</i>	
Nikolaj Björner	10, 21, 24
<i>(XDegrees Inc., USA)</i>	
Thierry Boy de la Tour	6
<i>(IMAG, France)</i>	
Gerhard Brewka	19
<i>(Universität Leipzig, Germany)</i>	
Uwe Egly	6
<i>(Technische Universität Wien, Austria)</i>	
Enrico Franconi	23
<i>(University of Manchester, UK)</i>	
Uli Furbach	3
<i>(Universität Koblenz-Landau, Germany)</i>	
Jean Goubault-Larrecq	5, 16
<i>(DYADE, France)</i>	
Bernhard Gramlich	9, 14
<i>(Technische Universität Wien, Austria)</i>	
Philippe de Groote	3
<i>(LORIA, France)</i>	
Robert Harper	17
<i>(Carnegie Mellon University)</i>	
Mitch Harris	9
<i>(University of Illinois at Urbana/Champaign)</i>	
John Harrison	15
<i>(Intel Research, USA)</i>	
Ian Horrocks	23
<i>(University of Manchester, UK)</i>	

Jieh Hsiang	7
<i>(National Taiwan University, Taiwan)</i>	
Ullrich Hustadt	4
<i>(University of Liverpool, UK)</i>	
Michael Kohlhase	15
<i>(Universität des Saarlandes, Germany)</i>	
Konstantin Korovin	9
<i>(University of Manchester, UK)</i>	
Christoph Kreiz	22
<i>(Cornell University, USA)</i>	
Christopher Lynch	7
<i>(Clarkson University, USA)</i>	
Michael Maher	12
<i>(Griffith University, Australia)</i>	
David McAllester	2, 13, 14
<i>(AT&T Labs Research, USA)</i>	
Gregory Mints	4
<i>(Stanford University, USA)</i>	
Gopalan Nadathur	16
<i>(University of Chicago, USA)</i>	
Paliath Narendran	8
<i>(University at Albany, USA)</i>	
Hans de Nivelle	22
<i>(Max-Planck-Institut für Informatik, Germany)</i>	
Uwe Petermann	28
<i>(Hochschule für Technik, Wirtschaft und Kultur Leipzig, Germany)</i>	
Frank Pfenning	4
<i>(Carnegie Mellon University, USA)</i>	
David Plaisted	5
<i>(University of North Carolina at Chapel Hill, USA)</i>	
Maarten de Rijke	23
<i>(University of Amsterdam, the Netherlands)</i>	
Alan Robinson	1
<i>(USA)</i>	
Michaël Rusinowitch	7
<i>(INRIA Lorraine, France)</i>	
Donald Sannella	14, 17
<i>(University of Edinburgh, UK)</i>	
Ulrike Sattler	23
<i>(RWTH Aachen, Germany)</i>	

Renate Schmidt	4
<i>(University of Manchester, UK)</i>	
Manfred Schmidt-Schauss	8
<i>(Universität Frankfurt, Germany)</i>	
Klaus Schulz	8
<i>(Ludwig-Maximilians-Universität München, Germany)</i>	
Jörg Siekmann	1
<i>(DFKI, Germany)</i>	
Jan Smith	17
<i>(Chalmers University, Sweden)</i>	
Viorica Sofronie	20
<i>(DFKI, Germany)</i>	
Perdita Stevens	24
<i>(University of Edinburgh, UK)</i>	
Mirek Truszczyński	19
<i>(University of Kentucky, USA)</i>	
Andrei Voronkov	7
<i>(University of Manchester, UK)</i>	
Lincoln Wallen	22
<i>(Mathengine, UK)</i>	
Christoph Walter	13
<i>(Technische Universität Darmstadt, Germany)</i>	
Dongming Wang	11
<i>(Université Pierre et Marie Curie, France)</i>	
Hantao Zhang	11
<i>(University of Iowa, USA)</i>	

The Early History of Automated Deduction

DEDICATED TO THE MEMORY OF HAO WANG

Martin Davis

SECOND READERS: Peter Andrews, Wolfgang Bibel, Alan Robinson, and
Jörg Siekmann.

Contents

1	Presburger's Procedure	5
2	Newell, Shaw & Simon, and H. Gelernter	6
3	First-Order Logic	7
	Bibliography	12
	Index	15

With the ready availability of serious computer power, deductive reasoning, especially as embodied in mathematics, presented an ideal target for those interested in experimenting with computer programs that purported to implement the "higher" human faculties. This was because mathematical reasoning combines objectivity with creativity in a way difficult to find in other domains. For this endeavor, two paths presented themselves. One was to try to understand what people do when they create proofs and to write programs emulating that process. The other was to make use of the systematic work of the logicians in reducing logical reasoning to standard canonical forms on which algorithms could be based. Each path confronted daunting obstacles. The difficulty with the first approach was that available information about how creative mathematicians go about their business was and remains vague and anecdotal. On the other hand, the well-known unsolvability results of Church and Turing showed that the kind of algorithm on which a programmer might want to base a theorem-proving program simply did not exist. Moreover, it was all too obvious that an attempt to generate a proof of something non-trivial by beginning with the axioms of some logical system and systematically applying the rules of inference in all possible directions was sure to lead to a gigantic combinatorial explosion.

Each of these approaches has led to important and interesting work. Unfortunately, for many years the proponents of the two approaches saw themselves as opponents and engaged in polemics in which they largely spoke past each other. One problem was that whereas they appeared to be working on the same problems, they tended to differ not only in their approaches, but also in their fundamental goals. Those whose method was the emulation of the human mathematician tended to see their research as part of an effort to help understand human thought. Those who proposed to use the methods of mathematical logic tended to see the goal as the development of useful systems of automated deduction. Ultimately, the most successful developments incorporated insights deriving from both approaches.

For a brief account of the history of the developments in logic that provided the background for research in this field see [Davis 1983c]. An interesting account of the two approaches and their mutual interaction can be found in [MacKenzie 1995]. The volume [Siekmann and Wrightson 1983] is a useful anthology of the principal articles on automated deduction to appear in the years through 1966.

1. Presburger's Procedure

In 1929, M. Presburger had shown that the first-order theory of addition in the arithmetic of integers is decidable, that is he had provided an algorithm which would be able to determine for a given sentence of that language, whether or not it is true. In 1954, Martin Davis programmed this algorithm for the vacuum tube computer at the Institute for Advanced Study in Princeton. As was stated by Davis [1983c]

Since it is now known that Presburger's procedure has worse than exponential complexity, it is not surprising that this program did not perform very well. Its

great triumph was to prove that the sum of two even numbers is even.

2. Newell, Shaw & Simon, and H. Gelernter

The propositional calculus is the most elementary part of mathematical logic, dealing as it does with the connectives $\neg \wedge \vee \supset$. Its treatment constitutes Section A of Part I (37 pages) of Whitehead and Russell's *Principia Mathematica*, their monumental three volume effort purporting to demonstrate that all of mathematics can be viewed as a part of logic. Their treatment proceeds from five particular formulas, that may be called *axioms* or *primitive propositions* to which are applied the explicitly stated rule of *modus ponens* or *detachment*¹ and implicit rules permitting substitutions for propositional variables and replacing defined symbols using their definitions. Newell, Shaw and Simon set themselves the problem of producing a computer program that emulates the process by which a person might seek proofs in the propositional calculus of *Principia*. Although the formalism is simple enough so that such a program would be feasible, the process requires enough ingenuity that the problem was hardly trivial.

In Newell, Shaw and Simon's [1957] report on experiments with their "Logic Theory Machine," (developed around the same time as Davis's Presburger program) the authors are very explicit about their goals:

Our explorations . . . represent a step in a program of research . . . aimed at developing a theory . . . and applying [it] to such fields as computer programming and human learning and problem-solving

Although it would be difficult to claim that this work has helped very much with such an ambitious agenda, it did provide a paradigm employed by many theorem-provers developed later, and this was surely its lasting influence. Among the techniques made explicit were forward and backward chaining, the generation of useful subproblems, and seeking substitutions that produce desired matches.

The authors emphasize that their program is "heuristic" rather than "algorithmic," and this purported distinction has given rise to much dissension and confusion. In this context, "heuristic" seems to mean little more than the lack of a guarantee that the process will always work (given sufficient space and time). The algorithm they contrast with their own procedure is the "British Museum algorithm" by which all possible proofs are generated until one leading to the desired result is reached. Indeed, the authors seem to have been unaware that Post's proof of the completeness of the *Principia* propositional calculus using truth tables had, in effect supplied a simple algorithm by means of which a demonstration by truth tables could be converted into a proof in *Principia* [Post 1921].

Wang and Gao [1987] presented a Gentzen-style proof system for the propositional calculus designed for efficiency. Unlike the program of Newell et al, Wang's system is complete: for any input, processing eventually halts, yielding either a

¹Actually Whitehead and Russell's tendency to confuse object and meta-language led them to state this confusingly as "Anything implied by a true proposition is true." But this lapse is not important for the present discussion.

proof or a disproof. The simple examples that are explicitly listed in *Principia*, including those that stumped the Logic Theory Machine, were easily disposed of. Although Wang seems not to have quite understood that producing an efficient generator of proofs in the propositional calculus was not what the Logic Theory Machine designers were after, they did leave themselves open to Wang's criticism by giving the impression that the absurd British Museum algorithm was the only possible "non-heuristic" proof-generating system for the propositional calculus.

Like the propositional calculus, the elementary geometry of the plane can be specified by a formal system for which an algorithmic decision procedure is available. This is seen by introducing a coordinate system and relying on the reduction of geometry to algebra and Tarski's decision procedure for the algebra of the real numbers. However, unlike the case of truth table methods for the propositional calculus, this method is utterly unfeasible. Although theoretical confirmation of this did not come until much later, it was already apparent from Davis's experience with the much simpler Presburger procedure. Herbert Gelernter's Geometry Machine is very much in the spirit of Newell et al. A clue to Gelernter's orientation is provided by his statement [Gelernter 1959]:

... geometry provides illustrative material in treatises and experiments in human problem-solving. It was felt that we could exchange valuable insights with behavioral scientists ...

Technically, in addition to the repertoire of The Logic Machine (backward chaining, the use of subproblems), the geometry machine introduced two interesting innovations: the systematic use of symmetries to abbreviate proofs and the use of a coordinate system to simulate the carefully drawn diagram a student of geometry might employ. This last was used to tip off the prover to the fact that certain pairs of line segments and of angles "appeared" to be equal to one another, and thereby to guide the search for a proof.

3. First-Order Logic

Unlike the cases of propositional logic and elementary geometry, there is no general decision procedure for first-order logic. On the other hand, given appropriate axioms as premises, all mathematical reasoning can be expressed in first-order logic, and that is why so much attention has been paid to proof procedures for this domain. Investigations by Skolem and Herbrand in the 1920s and early 1930s provided the basic tools needed for theorem-proving programs for first-order logic [Davis 1983c].

In 1957 a five week Summer Institute for Symbolic Logic held at Cornell University was attended by almost every logician working in the United States. Many of the more theoretically inclined researchers from the nearby IBM facilities were also present; FORTRAN, a brand-new innovation in programming practice was unveiled. After discussions with Gelernter, the logician Abraham Robinson was led to give a short talk [Robinson 1957] in which he pointed to Skolem functions and "Herbrand's theorem" as useful tools for general purpose theorem-provers. He also made the provocative remark that the auxilliary points, lines, or circles "constructed" as

part of the solution to a geometry problem can be thought of as being elements of what is now called the Herbrand universe for the problem.

The first theorem-provers for first-order logic to be implemented based on Herbrand's theorem employed a completely unguided search of the Herbrand universe. Instead of using Skolem functions to deal with instantiations, variables were replaced by parameters; so the program had to provide a capability for keeping track of dependencies among these parameters. Tests for truth-functional satisfiability used either simple truth table calculations or expansion into disjunctive normal form. Not surprisingly, these programs were capable of proving only the simplest theorems. Among the first of these programs, that by Gilmore [1960] served as a particularly useful stimulus for further investigations.

In his later commentary, Prawitz [1983a] explained that the development of new proof procedures and completeness proofs for first-order logic together with the availability of computational resources tempted him to become involved in implementing such a procedure. He adopted a modified form of the method of semantic tableaux, and formulated his own high level algorithmic language in which the procedure could be written. The detailed implementation was accomplished by Prawitz, Prawitz and Voghera [1960]. Despite being based on an up-to-date underlying logical system, this program suffered from the same limitations as Gilmore's.

Martin Davis and Hilary Putnam noted that Gilmore's program failed on some rather simple examples because of its reliance on expansions into disjunctive normal form for satisfiability testing. This led them to the optimistic (and in retrospect rather naive) conclusion that the lack of effective methods for testing large formulas of the propositional calculus for satisfiability was the main obstacle to be surmounted. Although their interest in algorithms for what came to be known as the *satisfiability problem* was only because they wanted to use such methods as part of a proof procedure for first-order logic, they secured support from the National Security Agency, to spend the summer of 1958 working on this problem. In their unpublished report to the NSA [Davis and Putnam 1958], they emphasized the use of conjunctive normal form for satisfiability testing. The specific reduction methods whose use together have been linked to the names Davis-Putnam are all present in this report. These are:

1. The *one literal rule* also known as the *unit rule*.
2. The *affirmative-negative rule* also known as the *pure literal rule*.
3. The *rule for eliminating atomic formulas*
4. The *splitting rule*, called in the report, the *rule of case analysis*

The Davis-Putnam paper usually cited [Davis and Putnam 1960] was written a year later. The proposed procedure would accept as input a formula that had been preprocessed by first using Skolem functions to eliminate existential quantifiers and then expanding the formula into conjunctive normal form. Many theorem-provers (including some that have been very successful) have used this approach. Satisfiability testing was to be carried out using rules 1,2,3 above, and it was noted that an example that stumped Gilmore's program could easily be done by hand computation. When George Logemann and Donald Loveland attempted to implement

the program they found that the *rule for eliminating atomic formulas* (later called *ground resolution*) which replaced a formula

$$(p \vee A) \wedge (\neg p \vee B) \wedge C$$

by

$$(A \vee B) \wedge C$$

used too much RAM. So it was proposed to use instead the *splitting rule* which generates the pair of formulas

$$A \wedge C \quad B \wedge C$$

The idea was that a stack for formulas to be tested could be kept in external storage (in fact a tape drive) so that formulas in RAM never became too large.² Although testing for satisfiability was performed very efficiently, it soon became clear that no very interesting results could be obtained without first devising a method for avoiding the generation of spurious elements of the Herbrand universe [Davis, Logemann and Loveland 1962].

During the same years, Hao Wang was attempting to apply some of the more sophisticated work that had been done in proof theory and on solvable cases of Hilbert's Entscheidungsproblem to automatic deduction programs. He announced a computer program that proved all of the theorems (about 400) of Whitehead and Russell's *Principia Mathematica* of first-order logic with equality [Wang and Zhi 1998, Wang and Zhi 1998]. However, this apparently momentous achievement in automating deduction was (as Wang himself pointed out) possible only because all of these theorems can be brought into prenex form with the simple prefix $\forall \dots \exists \dots \exists$. Wang concluded that:

The most interesting lesson from these results is perhaps that even in a fairly rich domain, the theorems actually proved are mostly ones which call on a very small portion of the available resources of the domain. ([Wang 1963c] p. 32)

Prawitz's [1960] influential paper taught the growing automated deduction community that unnecessary terms in the Herbrand expansion could be avoided by using algorithms that did not generate elements of the Herbrand universe until needed. Most later progress was based on this key insight. Prawitz's procedure worked by obtaining expansions into disjunctive normal form before replacing variables by

²Unfortunately both procedures using rules 1,2, and 3 and procedures using rules 1,2, and 4 have been called the "Davis-Putnam procedure" in the literature; the first is generally considered for worst case analysis while it is the second that is ordinarily implemented.

Wolfgang Bibel has kindly pointed out to me that the "rule for eliminating atomic formulas" otherwise known as "ground resolution" was first proposed in A. Blake's dissertation in 1937 and (in its dual form) was also mentioned by W.V. Quine in 1955 under the name "consensus rule". For further information, see [Bibel 1993]. Otherwise, as far as I know, the other rules mentioned occurred for the first time in [Davis and Putnam 1958].

It should also be mentioned that rules 2 and 4 were found independently by Dunham, Fridsal and Sward [1959]. They emphasized that a program based on these rules performs very effectively without using "heuristic" devices.

elements of the Herbrand universe. The algorithm thus generated disjunctive normal forms of increasing length seeking one with the property that some substitution from the Herbrand universe would yield a truth-functionally unsatisfiable formula.³ Since this condition amounts to each disjunctive clause including a pair of literals of the form $\ell, \neg\ell$, it can be formulated as the need to satisfy a system of equations in the parameters of the expansion.⁴

Prawitz's procedure was a great improvement over what had been done previously because no spurious elements of the Herbrand universe were generated. Unfortunately, the huge expansions into disjunctive normal form that would be generated by all but the simplest problems made it clear that, at least as presented, this was still an unsatisfactory procedure. However, it contained the seminal idea of searching for substitutions that would transform pairs of literals into negations of one another. Moreover if existential quantifiers are eliminated in favor of Skolem functions at the outset, instead of systems of equations, one has the simple problem of *unifying* pairs of terms.

In his survey paper, Davis [1963] proposed

... a new kind of procedure which seeks to combine the virtues of the Prawitz procedure and those of the Davis-Putnam procedure.

The idea, also noted by Dunham and North [1962], was that by the "pure literal rule" from the Davis-Putnam procedure (Rule 2, above), substitutions can help to render a conjunctive set of disjunctive clauses unsatisfiable only if they succeed in transforming a literal from one of these clauses into the negation of a literal in another clause. A theorem-proving program based on these ideas was written by D. McIlroy at Bell Laboratories and was improved and corrected by Peter Hinman. The program included an implementation of the ordinary unification algorithm [Chinlund, Davis, Hinman and McIlroy 1964].

Merely the existence of this volume makes it abundantly clear that automated reasoning is a thriving field with a huge literature. The bimonthly publication *The Journal of Automated Reasoning* is devoted entirely to this field. If one event can be pinpointed as marking its emergence as a mature subject, it would be the publication [Robinson 1965*b*] in which J.A. Robinson announced the resolution principle. [Robinson 1965*b*] was Robinson's second paper in the area, and it is helpful in tracing his thought to begin with the first [Robinson 1963]. He began with the basic framework of Davis-Putnam: existential quantifiers eliminated in favor of Skolem functions and conjunctive normal form. He noted Prawitz's technique for avoiding spurious elements of the Herbrand universe and Davis's survey paper. Evidently Davis's sketch of his proposed procedure was insufficiently clear, and Robinson wrote:⁵

³This account is not quite accurate because in Prawitz's paper matters are expressed in terms of finding a proof rather than a refutation. So what he actually did is precisely the dual of what is stated above.

⁴As pointed out to the author by Gérard Huet, this same use of equations occurs already in Herbrand's [1930, p. 145] thesis.

⁵In the interest of clarity, the reference numbers in this quote were replaced by the numbers in the present bibliography corresponding to the same papers.

Davis [1963] has therefore proposed a way of exploiting Prawitz' powerful idea while avoiding Prawitz' disastrous use of normal forms—in much the same way that the techniques of Davis and Putnam [1960] avoid the use of normal forms which caused Gilmore's [1960] program to be unable even to solve [an easy problem]. From the few remarks at the end of [Davis 1963] it does not yet seem clear just how Davis will proceed, and one waits with great interest his further researches along these lines.

The rest of the paper has a number of interesting computer proofs generated by using the Davis-Putnam “one literal clause” rule, and, when that fails, requiring the user to pre-specify the elements of the Herbrand universe needed to obtain a proof. Finding these elements was conjectured to be “the really ‘creative’ part of the art of proof-construction.”

Robinson's method of resolution introduced in his highly influential [1965*b*] revolutionized the subject. Robinson found a single rule of inference, easily performable by computer, that was complete for first-order logic. Using resolution required no separate procedure for dealing with propositional calculus. Starting with the usual pre-processed conjunctive set of disjunctive clauses, Robinson's technique was to seek all possible “unifications” that would make it possible to express the set of clauses as

$$(\ell \vee A) \wedge (\neg \ell \vee B) \wedge C$$

where ℓ is a literal that doesn't occur in C . This yields the “resolvent”

$$(A \vee B) \wedge C$$

which after $(A \vee B)$ is “multiplied out” yields a new set of clauses that is unsatisfiable just in case the original set was. This was similar to Davis's proposal [Davis 1963, Chinlund et al. 1964], in seeking unifications that generate complementary literals. It differs not only in not requiring separate truth functional testing, but also in not requiring, as part of the input, specification of the number of instances of each clause to participate in the final proof. [Robinson 1965*b*] is striking for its combinatorial simplicity, as well as for the sheer mathematical elegance of the presentation. Unfortunately, as soon became apparent, the bare resolution method could easily produce many thousands of clauses without reaching a proof. Finding a proof using resolution becomes the problem of providing criteria for the order in which resolutions are to be sought. Early attempts to cut down the search space were Robinson's own elegant hyperresolution [Robinson 1965*a*], and the strategies of unit preference [Wos, Carson and Robinson 1964] and set of support [Wos, Robinson and Carson 1965].

The three decades since the first implementations of resolution have seen an outpouring of research devoted to automated reasoning systems. While some of the most successful are based on resolution, others have proceeded in different directions. For further information, the reader is referred to the other articles in this volume.

Bibliography

- BIBEL W. [1993], *Deduction. Automated Logic.*, Academic Press. with the assistance of Steffen Hölldobler.
- CHINLUND T. J., DAVIS M., HINMAN P. AND MCILROY M. [1964], Theorem proving by matching. Bell Laboratories.
- DAVIS M. [1957], A computer program for presburger's algorithm, in 'Summaries of Talks Presented at the Summer Institute for Symbolic Logic', Institute for Defense Analysis. 2nd edition, published in 1960. Reprinted as [Davis 1983a].
- DAVIS M. [1963], Eliminating the irrelevant from mechanical proofs, in 'Proc. Symp. Applied Math.', Vol. XV, pp. 15–30. Reprinted as [Davis 1983b].
- DAVIS M. [1983a], A computer program for Presburger's algorithm, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 41–48. Originally published as [Davis 1957].
- DAVIS M. [1983b], Eliminating the irrelevant from mechanical proofs, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer, pp. 315–330. Originally published as [Davis 1963].
- DAVIS M. [1983c], The prehistory and early history of automated deduction, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 1–28.
- DAVIS, M., ED. [1994], *Solvability, Provability, Definability: The Collected Works of Emil L. Post*, Birkhäuser.
- DAVIS M., LOGEMANN G. AND LOVELAND D. [1962], 'A machine program for theorem proving', *Communications of the ACM* 5(1962), 394–397. Reprinted as [Davis, Logemann and Loveland 1983].
- DAVIS M., LOGEMANN G. AND LOVELAND D. [1983], A machine program for theorem proving, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer, pp. 267–270. Originally published as [Davis et al. 1962].
- DAVIS M. AND PUTNAM H. [1958], Computational methods in the propositional calculus, unpublished report, Rensselaer Polytechnic Institute.
- DAVIS M. AND PUTNAM H. [1960], 'A computing procedure for quantification theory', *Journal of the ACM* 7(3), 201–215. Reprinted as [Davis and Putnam 1983].
- DAVIS M. AND PUTNAM H. [1983], A computing procedure for quantification theory, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 125–150. Originally published as [Davis and Putnam 1960].
- DUNHAM B., FRIDSAL R. AND SWARD G. [1959], A non-heuristic program for proving elementary logical theorems, in 'Proc. IFIP Congr.', pp. 282–285.
- DUNHAM B. AND NORTH J. [1962], Theorem testing by computer, in 'Symp. Math. Theory Machines', Brooklyn Poly. Inst., pp. 172–177. Reprinted as [Dunham and North 1983].
- DUNHAM B. AND NORTH J. [1983], Theorem testing by computer, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 271–275. Originally published as [Dunham and North 1962].
- GELERNTER H. [1959], Realization of a geometry-theorem proving machine, in 'Proc. Intern. Conf. on Inform. Processing', UNESCO House, pp. 273–282. Reprinted as [Gelernter 1983].
- GELERNTER H. [1983], Realization of a geometry-theorem proving machine, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 99–122. Originally published as [Gelernter 1959].
- GILMORE P. [1960], 'A proof method for quantification theory: its justification and realization', *IBM J. of Research and Development* 4, 28–35. Reprinted as [Gilmore 1983].
- GILMORE P. [1983], A proof method for quantification theory: its justification and realization, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 151–158. Originally published as [Gilmore 1960].

- HERBRAND J. [1930], *Recherches sur la théorie de la démonstration*, PhD thesis, University of Paris, Austin. Translated as [Herbrand 1971].
- HERBRAND J. [1971], Investigations in proof theory, in W. Goldfarb, ed., 'Jacques Herbrand—Logical Writings', Harvard University Press. Originally published as [Herbrand 1930].
- MACKENZIE D. [1995], 'The automation of proof: an historical and sociological exploration', *IEEE Annals of the History of Computing* 17(3), 7–29.
URL: <http://dream.dai.ed.ac.uk/papers/donald/donald.html>
- NEWELL A., SHAW J. AND SIMON H. [1957], Empirical explorations with the logic theory machine, in 'Proc. West. Joint Comp. Conf.', pp. 218–239. Reprinted as [Newell, Shaw and Simon 1983].
- NEWELL A., SHAW J. AND SIMON H. [1983], Empirical explorations with the logic theory machine, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 49–73. Originally appeared as [Newell et al. 1957].
- POST E. [1921], 'Introduction to a general theory of elementary propositions', *American Journal of Mathematics* 43, 163–185. reprinted as [Post 1967] and in [Davis 1994].
- POST E. [1967], Introduction to a general theory of elementary propositions, in J. van Heijenoort, ed., 'From Frege to Gödel: a Source Book in Mathematical Logic, 1879–1931', Harvard University Press, Cambridge, MA, pp. 264–283.
- PRAWITZ D. [1960], 'An improved proof procedure', *Theoria* 26(2), 102–139. Reprinted as [Prawitz 1983].
- PRAWITZ D. [1983a], Comments on [Prawitz 1960] and [Prawitz et al. 1960], in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 159–161, 200–201.
- PRAWITZ D. [1983b], An improved proof procedure, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 162–199. Originally published as [Prawitz 1960].
- PRAWITZ D., PRAWITZ H. AND VOGHERA N. [1960], 'A mechanical proof procedure and its realization in an electronic computer', *Journal of the ACM* 7(2), 102–128. Reprinted as [Prawitz, Prawitz and Voghera 1983].
- PRAWITZ D., PRAWITZ H. AND VOGHERA N. [1983], A mechanical proof procedure and its realization in an electronic computer, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 202–228. Originally published as [Prawitz et al. 1960].
- ROBINSON A. [1957], Proving theorems, as done by man, machine and logician, in 'Summaries of Talks Presented at the Summer Institute for Symbolic Logic', Institute for Defense Analysis. 2nd edition, published in 1960. Reprinted as [Robinson 1983a].
- ROBINSON A. [1983a], Proving theorems, as done by man, machine and logician, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer, pp. 74–76. Originally published as [Robinson 1957].
- ROBINSON J. [1963], 'Theorem-proving on the computer', *Journal of the ACM* 10, 163–174. Reprinted as [Robinson 1983d].
- ROBINSON J. [1965a], 'Automatic deduction with hyper-resolution', *International Journal of Computer Mathematics* 1, 227–234. Reprinted as [Robinson 1983b].
- ROBINSON J. [1965b], 'A machine-oriented logic based on the resolution principle', *Journal of the ACM* 12(1), 23–41. Reprinted as [Robinson 1983c].
- ROBINSON J. [1983b], Automatic deduction with hyperresolution, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer, pp. 416–423. Originally published as [Robinson 1965a].
- ROBINSON J. [1983c], A machine oriented logic based on the resolution principle, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1965, Springer, pp. 397–415. Originally published as [Robinson 1965b].

- ROBINSON J. [1983d], Theorem-proving on the computer, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer, pp. 372–383. Originally published as [Robinson 1963].
- SIEKMANN, J. AND WRIGHTSON, G., EDS [1983], *Automation of Reasoning. Classical Papers on Computational Logic, 1957-1966*, Vol. I,II, Springer Verlag.
- WANG H. [1960a], 'Proving theorems by pattern recognition I', *Communications of the ACM* 3(1960), 220–234. Reprinted as [Wang 1983a].
- WANG H. [1960b], 'Towards mechanical mathematics', *IBM J. of Research and Development* 4, 2–22. Reprinted as [Wang 1983b].
- WANG H. [1961], 'Proving theorems by pattern recognition II', *Bell Syst. Tech. J.* 40, 1–41.
- WANG H. [1963a], Mechanical mathematics and inferential analysis, in Braffort and Hirschberg, eds, 'Computer Programming and Formal Systems', North-Holland, pp. 1–20.
- WANG H. [1963b], Mechanical mathematics and inferential analysis, in Braffort and Hirschberg, eds, 'Computer Programming and Formal Systems', North-Holland, pp. 1–20.
- WANG H. [1963c], The mechanization of mathematical arguments, in 'Proc. Symp. in Applied Math.', Vol. XV, pp. 31–40.
- WANG H. [1983a], Proving theorems by pattern recognition I, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1960, Springer Verlag, pp. 229–243. Originally published as [Wang and Hu 1987].
- WANG H. [1983b], Towards mechanical mathematics, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 244–264. Originally published as [Wang and Gao 1987].
- WOS L., CARSON D. AND ROBINSON G. [1964], The unit preference strategy in theorem proving, in 'IFIPS 1964 Fall Joint Comp. Conf.', Vol. 26, pp. 616–621. Reprinted in [Wos, Carson and Robinson 1983].
- WOS L., CARSON D. AND ROBINSON G. [1983], The unit preference strategy in theorem proving, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer, pp. 387–393. Originally appeared as [Wos et al. 1964].
- WOS L., ROBINSON G. AND CARSON D. [1965], 'Efficiency and completeness of the set of support strategy in theorem proving', *Journal of the ACM* 12, 536–541. Reprinted in [Wos, Robinson and Carson 1983].
- WOS L., ROBINSON G. AND CARSON D. [1983], Efficiency and completeness of the set of support strategy in theorem proving, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer, pp. 484–489. Originally appeared as [Wos et al. 1965].

Index

A

affirmative-negative rule 8

B

Bibel, Wolfgang 9

Blake, A. 9

C

chaining 6, 7

Church, Alonzo 5

conjunctive normal form 8

D

Davis, Martin 5, 8, 10–12

Davis-Putnam procedure 8–10

disjunctive normal form 8, 9

Dunham and North 10

Dunham, Fridsal, and Sward 9

E

Entscheidungsproblem 9

F

first-order logic 7

G

Gérard Huet 10

Gelernter, H. 7

Geometry Machine 7

Gilmore, P.C. 8

ground resolution 9

H

Herbrand universe 8–11

Herbrand's theorem 7, 8

Herbrand, J. 7

Hinman, Peter 10

hyperresolution 11

L

Logemann, George 8

Logic Theory Machine 6

Loveland, Donald 8

M

McIlroy, D. 10

N

Newell, Shaw and Simon 6

O

one literal rule 8, 11

P

Post, E.L. 6

Prawitz, Dag 8–10

Prawitz, Håkan 8

Presburger, M. 5

Principia Mathematica 6, 9

propositional calculus 6

pure literal rule 8, 10

Putnam, Hilary 8

Q

Quine, W.V. 9

R

resolution 10, 11

Robinson, Abraham 7

Robinson, J.A. 10, 11

rule for eliminating atomic formulas .. 8, 9

rule of case analysis 8

Russell, Bertrand 6

S

set of support 11

Skolem functions 7, 8, 10

Skolem, T. 7

splitting rule 8, 9

subproblems 6, 7

T

Tarski, Alfred 7

Turing, Alan 5

U

unification 10, 11

unit preference 11

unit rule 8

V

Voghera, Neri 8

W

Wang, Hao 6, 7, 9

Whitehead, A. N. 6

Resolution Theorem Proving

Leo Bachmair

Harald Ganzinger

SECOND READERS: David McAllester and Christopher Lynch.

Contents

1	Introduction	21
2	Preliminaries	22
2.1	Formulas and Clauses	22
2.2	Herbrand Interpretations	24
2.3	Rewrite Systems	24
2.4	Refutational Theorem Proving	25
2.5	Orderings	26
3	Standard Resolution	28
4	A Framework for Saturation-Based Theorem Proving	34
4.1	Theorem Proving Processes	35
4.2	Counterexample-Reducing Inference Systems	38
4.3	A Simple Resolution Prover for First-Order Clauses	41
5	General Resolution	46
5.1	Selection Functions	46
5.2	General Ordered Resolution	48
5.3	Refutational Completeness	49
5.4	Applications of Standard Redundancy	50
6	Basic Resolution Strategies	59
6.1	Binary Resolution	59
6.2	Hyper-Resolution	60
6.3	Boolean Ring-Based Methods	61
7	Refined Techniques for Defining Orderings and Selection Functions	66
7.1	Renaming and Semantic Resolution	66
7.2	Resolution with Free Selection	68
7.3	Conservative Extensions	69
7.4	Lock Resolution	71
7.5	The Inverse Method	76
7.6	Ordered Theory Resolution	78
8	Global Theorem Proving Methods	84
8.1	The Davis-Putnam Method	84
8.2	Saturated Semantic Tableaux	86

HANDBOOK OF AUTOMATED REASONING

Edited by Alan Robinson and Andrei Voronkov

© 2001 Elsevier Science Publishers B.V. All rights reserved

8.3	Model Elimination	87
9	First-Order Resolution Methods	89
9.1	First-Order Sequents	89
9.2	Lifting of Ordering Constraints	89
9.3	Constrained Formulas	90
9.4	Resolution Modulo an Equational Theory	91
10	Effective Saturation of First-Order Theories	91
10.1	Decision Procedures Based on Resolution	91
10.2	Automated Complexity Analysis	92
11	Concluding Remarks	93
	Bibliography	94
	Index	98

1. Introduction

Saturation-based theorem proving in its modern form was invented by Robinson [1965b] when he introduced the resolution calculus, the essence of which can be described by two inference rules:

(Binary) Resolution

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma}$$

where σ is the most general unifier of the atomic formulas A and B ;

(Positive) Factoring

$$\frac{C \vee A \vee B}{(C \vee A)\sigma}$$

where σ is the most general unifier of the atomic formulas A and B .

Resolution is a refutationally complete theorem proving method: a contradiction (i.e., the empty clause) can be deduced from any unsatisfiable set of clauses. The search for a contradiction proceeds by saturating the given clause set, that is, systematically (and exhaustively) applying all inference rules.

Resolution on ground clauses is a version of the cut rule restricted to atomic formulas, whereas factoring is an instance of contraction.¹ In fact, the refutational completeness of resolution can be derived from the completeness of the (propositional) sequent calculus; and Herbrand's theorem, which states that for any unsatisfiable set of non-ground clauses there is a finite set of ground instances that is propositionally unsatisfiable, establishes a link between propositional clauses and general clauses with variables. A key in the "lifting" argument is the existence (and uniqueness) of a most general unifier for any two unifiable atoms or terms.

But resolution is not primarily a method for deciding the unsatisfiability of propositional formulas (the procedure by Davis and Putnam [1960], for instance, is better suited for that purpose). Its main advantage over other early theorem proving methods, such as Gilmore's algorithm [1960], is that unification, as a selection mechanism for inferences, provides an effective way of interleaving the two processes: (i) the identification of suitable (ground) instances of clauses and (ii) a demonstration of their unsatisfiability.

In this chapter we describe the theoretical concepts and results that form the basis of state-of-the-art automated theorem provers based on resolution and refinements thereof. After presenting some preliminary material in Section 2, we explain, in Section 3, the main ingredients of resolution calculi — orderings and selection — and the main theoretical concepts — candidate models and reduction of counterexamples — which we use for obtaining completeness results for these calculi. In

¹Some textbooks (e.g., Gallier [1986]) take a different perspective and present resolution as a macro inference in the cut-free sequent calculus. However, when the clauses to be refuted are viewed as additional non-logical axioms, cuts can not be eliminated, but may be restricted to analytic cuts—the resolution inferences.

Section 4 we describe a general framework for modeling theorem proving processes that involve both simplification and search. This formalism is applied to an extended example of a resolution-based theorem prover with simplification in Section 4.3. In Sections 5, 6, and 7 we present, and refine, a general resolution calculus for general clauses, i.e., clauses which are (disjunctive) multisets of arbitrary quantifier-free formulae. This calculus serves mainly as a theoretical concept from which most of the major resolution-based calculi (binary resolution, positive resolution, semantic resolution, hyper-resolution, non-clausal resolution, theory resolution, the inverse method, Boolean ring-based methods) can be derived as special cases. Viewing the specialized calculi as different instances of one general inference system also sheds light on their mutual relationships.

For simplicity the presentation of most inference systems is initially given for variable-free formulas only. In Section 9 we briefly discuss techniques for lifting our results to the general level of formulas with variables.

We will also indicate how the theory of resolution can be applied to obtain refinements of tableau-based theorem proving methods by arguing, in Section 8, that the notion of a “closed tableau” can be generalized to that of a “saturated tableau” in which all paths are saturated, up to redundancy, by ordered resolution. In Section 10 we discuss the role of resolution-based methods, not only for refutational theorem proving, but also as a tool for analyzing and compiling presentations of logical theories. It will be briefly explained how saturation may help in automatically generating decision procedures for a theory so that certain complexity bounds for the entailment problem for the theory can be guaranteed. We also show that saturation may be used as a tool for generating variants of resolution calculi that are specifically tailored to certain theories such as orderings or congruences. This provides new insights on how to compute with large, but structured theories.

A distinctive feature of our presentation is that we not only place local restrictions on resolution inferences, but via an abstract notion of redundancy also provide a framework in which global restrictions on the proof search can be expressed and justified by means of simplification and elimination. This allows us to answer questions about the compatibility, or incompatibility, of such techniques as tautology elimination, subsumption, normal form transformation, or reduction with various types of resolution calculi.

2. Preliminaries

2.1. Formulas and Clauses

We consider quantifier-free first-order formulas built from variables, function symbols, predicate symbols and logical connectives. We will deal with the logical symbols \top (verum), \perp (falsum), \neg (negation), \vee (disjunction), \wedge (conjunction), \supset (implication), \oplus (exclusive disjunction), and \equiv (equivalence), though our results apply to other connectives as well. A *term* is either a variable or an expression $f(t_1, \dots, t_n)$, where f is an n -ary function symbol and t_1, \dots, t_n are terms.

An *atomic formula* (or *atom*) is an expression $P(t_1, \dots, t_n)$, where P is a predicate symbol of arity n and t_1, \dots, t_n are terms. A predicate symbol of arity 0 is called a *propositional constant*. A *literal* is an expression A (a *positive literal*) or $\neg A$ (a *negative literal*), where A is an atomic formula. Two literals A and $\neg A$ are said to be *complementary*.

Calculi for automated deduction are often described in terms of constructs that represent formulas, but abstract from certain non-essential aspects of the syntax or encode additional structural information. For example, multiple disjunctions or conjunctions may be conveniently represented as sequences (or multisets), due to the associativity (and commutativity) property of these connectives.

A *multiset* over a set S is a function Σ from S to the natural numbers. Intuitively, $\Sigma(x)$ specifies the number of occurrences of x in Σ . We say that x is an *element* of Σ if $\Sigma(x) > 0$. A set may be thought of as a multiset Σ for which $\Sigma(x)$ is 0 or 1, for all x . A multiset Σ is *finite* if $\Sigma(x) = 0$ for all but finitely many x . The *union* and *intersection* of multisets are defined by the identities $\Sigma_1 \cup \Sigma_2(x) = \Sigma_1(x) + \Sigma_2(x)$ and $\Sigma_1 \cap \Sigma_2(x) = \min(\Sigma_1(x), \Sigma_2(x))$. If Σ is a multiset and S a set, we write $\Sigma \subseteq S$ to indicate that every element of (the multiset) Σ is an element of (the set) S , and use $\Sigma \setminus S$ to denote the multiset Σ' for which $\Sigma'(x) = 0$ for any x in S , and $\Sigma'(x) = \Sigma(x)$, otherwise. We often use sequences or set-like notation to denote multisets and write, for instance, Σ, Δ instead of $\Sigma \cup \Delta$, or Σ, A instead of $\Sigma \cup \{A\}$. For example, by $\neg A, B, B$ we denote the multiset Σ over formulas for which $\Sigma(\neg A) = 1$, $\Sigma(B) = 2$, and $\Sigma(F) = 0$, for all other formulas F .

A finite multiset of formulas may either be interpreted as the disjunction or as the conjunction of its elements. We will interpret multisets as disjunctions and speak of *general clauses*. The empty multiset represents the constant \perp . If (F_1, \dots, F_n) is a general clause, we write $\neg(F_1, \dots, F_n)$ to denote the formula $\neg F_1 \wedge \dots \wedge \neg F_n$. If all the elements in a general clause are literals, it is called a *standard clause*. Standard clauses are usually written as disjunctions, $L_1 \vee L_2 \vee \dots \vee L_n$. We use calligraphic letters \mathcal{C}, \mathcal{D} , and \mathcal{G} to denote general clauses, and capital letters C and D to denote standard clauses.

On rare occasions we will interpret multisets as conjunctions, in which case we speak of *dual* (general or standard) clauses. Note that the empty dual clause represents the constant \top .

We write $E[E']_p$ to denote the expression that is obtained from replacing the subexpression at position p in E by E' .² Thus, $E[E']_p$ is an expression that contains E' as a subexpression (at position p). The position p may be omitted if it is clear from the context. Sometimes we use p to denote a set of positions in E , in which case replacement is meant to take place at all the positions in p simultaneously. We also write $E(E')$ to indicate that E contains at least one occurrence of E' as a subexpression. By $E[E'/E'']$ or, if E' is known from the context, simply $E(E'')$, we denote the result of simultaneously replacing *all* occurrences of E' in E by E'' .

Variable-free expressions are called *ground* or *closed*. When we wish to emphasize that an expression may contain variables we also speak of a *first-order expression*.

²Positions may be represented in Dewey decimal notation, as sequences of non-negative integers.

2.2. Herbrand Interpretations

A (*Herbrand*) *interpretation* is a set of ground atoms. A ground atom A is said to be *true* in an (Herbrand) interpretation I if $A \in I$, and *false* otherwise. The logical connectives are interpreted in the usual way. The constant \top is true in all interpretations, whereas \perp is false in all interpretations. A conjunction $A \wedge B$ is true in I if both A and B are true in I ; a disjunction $A \vee B$ is true if at least one of A and B is true; etc. The truth value of a formula depends only on the truth values assigned to its atomic formulas. A general clause (F_1, \dots, F_n) is true in I if at least one of the formulas F_i is true in I (whereas a dual clause (F_1, \dots, F_n) is true in I if all of the formulas F_i are true in I).

An interpretation I is called a *model* of an expression E if E is true in I ; and a model of a set of expressions N , if it is a model of all expressions in N . An expression or a set of expressions is called *satisfiable*, or *consistent*, if it has a model; and *unsatisfiable*, or *inconsistent*, otherwise. An expression that is true in all interpretations is said to be *valid*, or a *tautology*. We also say that E' is a *logical consequence* of E , or *logically follows* from E , or that E *logically implies* E' (written $E \models E'$), if E' is true in all models of E . Two expressions E and E'' are said to be (*logically*) *equivalent*, written $E \simeq E''$, if, and only if, they have the same truth value in each interpretation (i.e., are either both true or both false). By a *contradiction* we mean an inconsistent expression that contains only the constants \top and \perp and logical connectives, but no function or predicate symbols. For example, \perp and $\top \supset \perp$ are contradictions, whereas $A \wedge \neg A$ is inconsistent, but not a contradiction in this sense.

2.3. Rewrite Systems

Rewrite systems are a basic tool for describing a variety of theorem proving techniques. We use the letters α, β, \dots to denote variables ranging over formulas. Syntactically, these “meta-variables” are treated like propositional constants.

A *substitution* is a mapping defined on variables, where variables denoting terms are mapped to terms and variables denoting formulas, to formulas. By $E\sigma$ we denote the result of applying the substitution σ to an expression E and call $E\sigma$ an *instance* of E . If $E\sigma$ is *ground* (i.e., contains no variables), we speak of a *ground instance* of E . Composition of substitutions is denoted by juxtaposition. Thus, if τ and ρ are substitutions, then $x\tau\rho = (x\tau)\rho$, for all variables x .

An *equivalence (relation)* is a reflexive, transitive, symmetric binary relation. For example, logical equivalence is indeed an equivalence relation. A binary relation \Rightarrow on expressions with variables is called a *rewrite relation* if $E' \Rightarrow E''$ implies $E[E'] \Rightarrow E[E'']$, for all expressions E , E' and E'' . If \Rightarrow is a binary relation, we denote by \Rightarrow^+ its transitive closure; by \Rightarrow^* its transitive-reflexive closure; by \Leftrightarrow its symmetric closure; and by \Leftrightarrow^* its transitive-reflexive-symmetric closure.

A *rewrite system* is a binary relation on expressions with variables, the elements of which are called *rewrite rules* and written $E \Rightarrow E'$. (We occasionally speak of a

two-way rewrite rule, and write $E \Leftrightarrow E'$, if a rewrite system contains both $E \Rightarrow E'$ and $E' \Rightarrow E$.) If R is a rewrite system, we denote by \Rightarrow_R the smallest rewrite relation that contains all instances $E\sigma \Rightarrow E'\sigma$ of rules in R . We say that E can be *rewritten to E'* by R , if $E \Rightarrow_R E'$. A rewrite relation defined on formulas can be extended to clauses as follows: $C \Rightarrow_R C'$ if C can be written as D, F and C' as D, F' , for some clause D and formulas F and F' with $F \Rightarrow_R F'$.

Expressions that can not be rewritten are said to be in *normal form*. We write $E \Rightarrow_R^! E'$ to indicate that $E \Rightarrow_R^* E'$ and E' is in normal form. We say that R *terminates* if there is no infinite sequence $E_0 \Rightarrow_R E_1 \Rightarrow_R \dots$ of rewrite steps. If R terminates, then every formula can be rewritten to a normal form (in zero or more steps).

If R and S are rewrite systems, we denote by R/S (R modulo S) the rewrite system consisting of all rules $E \Rightarrow E'$, such that $E \Leftrightarrow_S^* G \Rightarrow_R G' \Leftrightarrow_S^* E'$, for some expressions G and G' .

2.4. Refutational Theorem Proving

Theorem provers are procedures that can be used to check whether a given formula F (the “goal”) is a logical consequence of a set of formulas N (the “theory”). Refutational theorem provers deal with the equivalent problem of showing that the set $N \cup \{\neg F\}$ is inconsistent. The inconsistency of a set N may be established either by a semantic analysis or by providing a formal proof of \perp from N , where proofs are traces of deductive inferences defined by a collection of inference rules.

For our purposes, an *inference rule* is an $n + 1$ -ary relation on general clauses.³ The elements of such a relation are usually written as

$$\frac{C_1 \quad \dots \quad C_n}{C}$$

and called *inferences*. The clauses C_1, \dots, C_n are called the *premises*, and C the *conclusion*, of the inference. An *inference system* Γ is a collection of inference rules.

If I is an inference or a set of inferences we denote by $C(I)$ its conclusion or the set of their conclusions. We also speak of an inference *from* a set of clauses N if all premises are elements of N , and denote by $\Gamma(N)$ the set of all inferences by Γ from N .

An inference is said to be *sound* if its conclusion is a logical consequence of its premises, $C_1, \dots, C_n \models C$. Soundness is often the minimal requirement expected in an inference system, but in refutational theorem proving it is sufficient that inferences preserve consistency. We call an inference system Γ *consistency-preserving* if for all sets of clauses N , the set $N \cup C(\Gamma(N))$ is consistent whenever N is consistent. A sound inference system is consistency-preserving, but the converse is not true in general. We will only consider consistency-preserving inference systems.

³In refutational theorem proving inferences without premises, or axioms, are of little use, so that we always have $n \geq 1$.

For the inference systems that we study in this chapter, the order of the premises in an inference is relevant, and we view inferences as mechanisms whereby a distinguished premise (the *main premise*) is “reduced” to the conclusion in the context of the other premises (the *side premises*). Unless specified otherwise, the last premise of an inference is the main premise, and the other premises, if any, the side premises.

A *proof* of a clause C from a set of clauses N with respect to an inference system Γ is a sequence of clauses C_1, \dots, C_m , such that $C = C_m$ and each clause C_i is either an element of N or else the conclusion of an inference by Γ from $N \cup \{C_1, \dots, C_{i-1}\}$. The clauses in N are also called *assumptions*. We write $N \vdash_{\Gamma} C$ if there exists a proof of C from N by Γ . If C is a contradiction we speak of a *refutation* of N .

An inference system Γ is said to be *refutationally complete* if there is a refutation by Γ from any unsatisfiable set of clauses N . A set of clauses N is called *saturated* with respect to Γ if the conclusion of any inference by Γ from N is an element of N . If an inference system Γ is refutationally complete and a set N is saturated with respect to Γ , then N is either satisfiable or contains a contradiction.

2.5. Orderings

Many theorem proving calculi employ orderings of one kind or another to obtain an approximate measure of the progress of a derivation towards a particular goal.

A (strict) *partial ordering* is a transitive and irreflexive binary relation; a *quasi-ordering* a reflexive and transitive binary relation. If \succ is a strict ordering, its reflexive closure \succeq is defined by: $x \succeq y$ if $x \succ y$ or $x = y$. A strict ordering is said to be *total* (on a subset S of the domain) if for any two distinct elements x and y (in S) we have either $x \succ y$ or $y \succ x$. The reflexive closure of a strict ordering is a quasi-ordering. On the other hand, if \succeq is a quasi-ordering, then its *strict part* \succ defined by: $x \succ y$ if $x \succeq y$ but not $y \succeq x$, is a strict ordering. We may also define an equivalence relation by: $x \sim y$ if $x \succeq y$ and $y \succeq x$. We say that an ordering \succ' *extends* \succ if the latter is a subset of the former, i.e., $x \succ' y$ whenever $x \succ y$.

For example, we may compare formulas by their size and define either a strict ordering: $F \succ G$ if G is shorter (as a string) than F ; or a quasi-ordering: $F \succeq G$ if F is not shorter than G . If F and G are of the same length, we have $F \succeq G$ and $G \succeq F$, but neither $F \succ G$ nor $G \succ F$. This ordering extends the subformula relation. The example also shows that the reflexive closure of the strict part of a quasi-ordering may be different from the quasi-ordering.

A strict ordering \succ is said to be *well-founded* if there is no infinite descending chain $x_1 \succ x_2 \succ \dots$ of elements. If \succ is a well-founded ordering on a set S , a property P is true for all elements of S whenever the implication “if $P(y)$, for each y in S such that $x \succ y$, then $P(x)$ ” holds for each x in S . This proof principle is called *Noetherian* or *well-founded induction*. A quasi-ordering is well-founded if its strict part is well-founded.

We say that an ordering \succ has the *subterm property* if $E[E'] \succ E'$, for all expressions E and proper subexpressions E' of E . A *rewrite ordering* is an ordering that is also a rewrite relation; a *reduction ordering*, a well-founded rewrite ordering;

and a *simplification ordering*, a reduction ordering with the subterm property. Note that a rewrite system R terminates if, and only if, there exists a reduction ordering \succ such that $E\sigma \succ E'\sigma$, for each rule $E \Rightarrow E'$ in R and each substitution σ . Reduction orderings must be compatible with the subterm ordering in that $E' \not\succeq E[E']$ for all expressions E and subexpressions E' of E . Thus, if E' and $E[E']$ are comparable with respect to a reduction ordering \succ , then $E[E'] \succeq E'$.

In theorem proving applications orderings are often defined with respect to the tree structure of terms and formulas. Let \succ be an ordering, called a *precedence*, on the given set of (function, predicate and logical) symbols. The corresponding *lexicographic path ordering* \succ_{lpo} is defined by:

$s = f(s_1, \dots, s_m) \succ_{lpo} g(t_1, \dots, t_n) = t$ if and only if

- (i) $f \succ g$ and $s \succ_{lpo} t_i$, for all i with $1 \leq i \leq n$; or
- (ii) $f = g$ and, for some j , we have $(s_1, \dots, s_{j-1}) = (t_1, \dots, t_{j-1})$, $s_j \succ_{lpo} t_j$, and $s \succ_{lpo} t_k$, for all k with $j < k \leq n$; or
- (iii) $s_j \succ_{lpo} t$, for some j with $1 \leq j \leq m$.

If the precedence is well-founded, the lexicographic path ordering is a simplification ordering. It is total on closed expressions whenever the precedence is total. For a survey on termination orderings see [Dershowitz 1987].

Any ordering on a set N can be extended to an ordering \succ_{mul} on finite multisets over N as follows: $\Sigma_1 \succ_{mul} \Sigma_2$ if (i) $\Sigma_1 \neq \Sigma_2$ and (ii) whenever $\Sigma_2(x) > \Sigma_1(x)$ then $\Sigma_1(y) > \Sigma_2(y)$, for some y such that $y \succ x$. (Here $>$ denotes the standard "greater-than" relation on the natural numbers.) Given a multiset, any smaller multiset is obtained by (repeatedly) replacing an element by zero or more occurrences of smaller elements. If an ordering \succ is total (resp., well-founded), so is its multiset extension. For example, if we order formulas by their size, then $P(f(a)) \succ Q(a)$ and $\{P(f(a))\} \succ_{mul} \{P(a), Q(a)\}$. For simplicity we often use the same symbol to denote both an ordering and its multiset extension.

If Σ is a multiset over M and x an element in M , x is said to be [strictly] *maximal* with respect to Σ if there is no element y in Σ such that $y \succ x$ [$y \succeq x$]. Since clauses are multisets, we may thus speak of the maximal formula in a general clauses, or the maximal literal in a standard clause. In addition, we say that A is a *maximal atom* of a clause C if A occurs in C and there is no atom B in C with $B \succ A$.

A reduction ordering \succ that is total on closed formulas is called *admissible* if (i) $A \succ \top$ and $A \succ \perp$, for all atoms A ; and (ii) $F \succ G$, whenever for all atoms B in G there exists an atom A in F such that $A \succ B$. An ordering \succ on ground clauses is *admissible* if it is the multiset extension of an admissible ordering on formulas. If an ordering on clauses is admissible, then it is well-founded and total on ground clauses.

A lexicographic path ordering over a total precedence is admissible if predicate symbols have higher precedence than logical symbols and the constants \top and \perp are smaller than the other logical symbols. If one regards ground atoms as constants, and uses a well-founded ordering \succ on atoms as a precedence, extended by $A \succ \equiv \succ \supset \succ \neg \succ \vee \succ \wedge \succ \top \succ \perp$, then the corresponding lexicographic path ordering is admissible. In other words, a well-founded ordering on ground atoms can always be extended to an admissible ordering on clauses.

3. Standard Resolution

Saturation-based theorem proving refers to a process in which two levels of data structures are manipulated. At the *level of deduction* new formulas are derived from given ones by applying specified inference rules, with the ultimate goal of obtaining a contradiction. In addition, the current set of formulas is analyzed to identify the most promising inference rules to be applied next and to eliminate redundancies. We will formalize this second level in terms of *theorem proving derivations*. The set of retained formulas represents the logical information that is explicitly available to a prover at a specific point in time, and provides the information used to determine further steps in the theorem proving process. Practical experience with theorem provers has shown that powerful and efficient “global” techniques for simplification of the current collection of formulas, and especially the elimination of redundancies therein, are far more important to an acceptable performance than any “local” refinements of the deductive inferences at the formula level. But inference computation and redundancy elimination interact in subtle ways and one has to be careful that their integration is not counterproductive and that refutational completeness is preserved.

In this chapter, and in more detail in Chapter 5 we will investigate resolution-based theorem proving methods for standard variable-free clauses. A standard clause is a multiset of literals. We say that an atom A occurs *positively* in a clause C , if A is one of the literals of C , and occurs *negatively* if $\neg A$ is a literal in C .

Numerous versions of resolution for standard clauses have been proposed in the literature, most of which will be discussed in later chapters. The following variant, for ground clauses, combines factoring and resolution into a single inference rule.

Binary resolution with factoring

$$\frac{C \vee A \vee \dots \vee A \quad \neg A \vee D}{C \vee D}$$

We speak of a *resolution on A* and call the conclusion of the inference a *resolvent* of the two premises. The main premise of the inference is $\neg A \vee D$, while $C \vee A \vee \dots \vee A$ is the side premise. By B we denote the set of all inferences by binary resolution with factoring. The calculus B is used in Figure 1 to derive a contradiction from five given input clauses.

Resolution is a sound inference. Suppose I is an interpretation in which both premises are valid. The atomic formula A is either true or false in I . If A is true in I then D must be true in I , for otherwise the main premise would be false in I . Similarly, if $\neg A$ is true in I then C must be true in I . In either case, the resolvent will be true in I .

Binary resolution with factoring is also refutationally complete, which we prove by showing that any inconsistent set of clauses that is closed under B contains a contradiction. In its contrapositive form this statement asserts that any set of clauses that is closed, but contains no contradiction, has a model. We specify a Herbrand model by induction on a suitably chosen clause ordering.

(1)	$\neg A \vee B$	[input]
(2)	$\neg B \vee C$	[input]
(3)	$A \vee \neg C$	[input]
(4)	$A \vee B \vee C$	[input]
(5)	$\neg A \vee \neg B \vee \neg C$	[input]
(6)	$A \vee B \vee A$	[resolving on C in (4) and (3)]
(7)	$B \vee B$	[resolving on A in (6) and (1)]
(8)	C	[resolving on B in (7) and (2)]
(9)	A	[resolving on C in (8) and (3)]
(10)	$\neg B \vee \neg C$	[resolving on A in (9) and (5)]
(11)	$\neg C$	[resolving on B in (7) and (10)]
(12)	\perp	[resolving on C in (8) and (11)]

Figure 1: A sample refutation

Let \succ be a total admissible ordering on clauses. Given a set of clauses N , we use induction with respect to \succ to define for each clause C (not necessarily in N) a Herbrand interpretation I_C and a set ε_C as follows.

3.1. DEFINITION. Take I_C to be the set $\bigcup_{C \succ D} \varepsilon_D$. Furthermore, if C is a clause that

- (i) is contained in N ;
- (ii) is of the form $C' \vee A$, where A is the maximal literal in C ; and
- (iii) is false in I_C ;

then $\varepsilon_C = \{A\}$; otherwise, ε_C is the empty set.

We also say that C *produces* A , and call C a *productive clause*, if $\varepsilon_C = \{A\}$. I_C is called the *partial interpretation below* C . By the *partial interpretation at* C we mean the (possibly extended) set $I^C = I_C \cup \varepsilon_C$. Finally, by the *candidate model* for N , denoted by I_N^* or simply I_N , we mean the Herbrand interpretation $\bigcup_{C \in N} \varepsilon_C$.

The partial interpretation I_C is intended to be a model of the set N_C of those clauses in N that are smaller than C (with respect to the given clause ordering); whereas ε_C is meant to be a minimal extension of I_C that makes C true. Only clauses in N in which the maximal atom A is positive, can possibly be productive. We say that such a clause is *reductive* for A . Reductive clauses may be viewed as implications $\neg C \supset A$. The recursive evaluation of the condition $\neg C$ (via “negation as failure”) will eventually terminate. If the condition $\neg C$ evaluates to true and A is not yet contained in the corresponding partial interpretation, then extending the interpretation by A will make the reductive clause true, but does not affect the

truth values of the (smaller) clauses which were used to evaluate its condition $\neg C$.

The following example shows that the construction need not always yield a model of (non-productive clauses in) N .

3.2. EXAMPLE. Take an ordering $B_2 \succ A_2 \succ B_1 \succ A_1 \succ B_0 \succ A_0$ on atoms. The following table describes the various partial interpretations for clauses, listed in ascending order.

clause C	interpretation I_C	ε_C	remarks
$A_0 \vee B_0$	\emptyset	$\{B_0\}$	B_0 is maximal
$B_0 \vee A_1$	$\{B_0\}$	\emptyset	true in I_C
$\neg B_0 \vee A_1$	$\{B_0\}$	$\{A_1\}$	A_1 is maximal
$\neg B_0 \vee A_2 \vee B_1$	$\{B_0, A_1\}$	$\{A_2\}$	A_2 is maximal
$\neg B_0 \vee \neg A_2 \vee B_1$	$\{B_0, A_1, A_2\}$	\emptyset	B_1 not maximal
$\neg B_1 \vee B_2$	$\{B_0, A_1, A_2\}$	\emptyset	true in I_C

The next to last clause is false in the final interpretation $\{B_0, A_1, A_2\}$.

The following lemmas clarify key connections between candidate models and clauses.

3.3. LEMMA. *If C is productive, then C is true in I_N .*

PROOF. By the construction, if C produces an atom A then C is true in $I_C \cup \{A\}$. Since $I_C \cup \{A\} \subseteq I_N$, the clause C is also true in I_N . \square

3.4. LEMMA. *Let C and D be clauses such that $D \succeq C$. If C is true in I_D or I^D then C is also true in I_N and in all interpretations $I_{D'}$ and $I^{D'}$, where $D' \succeq D$.*

PROOF. First, observe that $I_D \subseteq I^D \subseteq I_{D'} \subseteq I^{D'} \subseteq I_N$, whenever $D' \succeq D$. If C contains a positive literal A that is true in I_D or I^D , then A is also true in I_N and in all interpretations $I_{D'}$ and $I^{D'}$, so that the assertion follows immediately. Otherwise, C must contain a negative literal $\neg A$, such that A is false in I_D . Suppose there is a clause D'' that is reductive for A . Then A is the maximal literal in D'' , but since admissible orderings are reduction orderings and total they satisfy the subterm property, so that $\neg A \succ A$, and hence $D \succeq C \succeq \neg A \succ D''$. This contradicts the fact that A is false in I_D . We conclude that A is false, and $\neg A$ and C are true, in I_N and in all interpretations $I_{D'}$ and $I^{D'}$. \square

We often use this lemma in its contrapositive form to infer that if a clause C is false some interpretation I_N or $I_{D'}$ or $I^{D'}$, and D is a clause with $D' \succeq D \succeq C$, then C is also false in I_D and in I^D .

3.5. LEMMA. *If C is a clause in N , the maximal literal of which is positive, then C is true in I_N .*

PROOF. If the maximal literal of a clause C in N is positive, then the clause is either productive or else is true in I_C . By the above lemmas, the clause is true in I_N . \square

3.6. LEMMA. *Let D and D' be clauses such that $D \succeq D'$ and either D' is a clause in N or else the maximal atom in D is strictly greater than the maximal atom in D' . If D' is false in I^D , then it is also false in I_N and in all interpretations I_C and I^C , where $C \succ D$.*

PROOF. Let D and D' be clauses as specified. Suppose D' is false in I^D , but true in I_N or in some interpretation I_C or I^C , where $C \succ D$. This is only possible if D' contains a positive occurrence of an atom A that is produced by some clause $C' \succ D$. But in that case A must be the maximal atom in D' , so that D' can not be in N , for otherwise it would not be false in I^D by the above lemmas. Therefore A must be strictly smaller than the maximal atom of D , which contradicts the assumption that C' is a productive clause for A with $C' \succ D$. \square

3.7. LEMMA. *If D is a clause in N and another clause C is true in all interpretations $I^{D'}$, where $D \succ D'$, then D is also true in I_D .*

PROOF. Let C and D be clauses as specified. If some atom that occurs positively in C is produced by a clause strictly smaller than D , then C is obviously true in I_D . Suppose, on the other hand, that none of the positive atoms in C , but all of the atoms with negative occurrences, are produced by clauses strictly smaller than D . Let A be the maximal atom that occurs negatively in C , and C' be the clause that produces A . Then C is already false in $I^{C'}$, contradicting our assumption. We may conclude that whenever none of the positive atoms in C is true in I_D , then some atom with a negative occurrence is false in I_D , making C true in I_D . \square

We have seen in the above example that non-productive clauses may be false in I_N . A clause that is false in an interpretation I is also called a *counterexample* for I . If a set N contains a counterexample for I_N , then it must also contain a minimal counterexample for I_N (with respect to the admissible clause ordering \succ). The following key result indicates that a minimal counterexample is either a contradiction or else can be reduced to an even smaller counterexample by resolution.

3.8. THEOREM. *Let N be a set of clauses not containing the empty clause and C be a minimal counterexample in N for I_N . Then there exists an inference by binary resolution with factoring from C such that*

- (i) *its conclusion is a counterexample for I_N and is smaller than C ; and*
- (ii) *C is its main premise and the side premise is a productive clause.*

PROOF. For simplicity let us denote I_N by I . By Lemma 3.3, productive clauses are true in I . Therefore, the minimal counterexample C must be a non-productive clause. Since N does not contain the empty clause, C must contain at least one literal. Let A be its maximal atom, which must occur negatively for C to be a

counterexample for I . Thus C can be written as $\neg A \vee C'$, where A is true in I . Let $D = D' \vee A \vee \dots \vee A$ be the clause that produces A , with a subclause D' that does not contain A . The clause D is reductive for A , so that all atoms in D' are strictly smaller than A . Moreover, D and, hence, D' are false in I_D . Therefore, resolution with the productive clause D as side premise and C as main premise yields a resolvent $D' \vee C'$ that is strictly smaller than C . By Lemma 3.6, D' is false in I , and consequently $D' \vee C'$ is false in I . In short, resolution yields a smaller counterexample than C . \square

3.9. THEOREM. *If N is saturated with respect to B and does not contain the empty clause then I_N is a model of N .*

PROOF. Theorem 3.8 indicates that the only counterexample that can not be reduced to an even smaller counterexample by resolution is the empty clause. For saturated sets N there are thus only two possibilities: the set either contains no counterexample (so that I_N is a model of N), or else contains the empty clause. \square

The theorem also implies that B is refutationally complete, for if the empty clause can not be derived, then N has a model.

3.10. COROLLARY. *The inference system B is refutationally complete.*

3.11. EXAMPLE. Continuing Example 3.2 we observe that $\neg B_0 \vee \neg A_2 \vee B_1$ is a smallest counterexample, with A_2 as maximal atom. The atom A_2 is produced by $\neg B_0 \vee A_2 \vee B_1$. Resolving the two clauses yields a smaller counterexample $\neg B_0 \vee B_1 \vee \neg B_0 \vee B_1$ to the initial candidate model. A modified model construction with this additional resolvent yields

clause C	interpretation I_C	ε_C	remarks
$A_0 \vee B_0$	\emptyset	$\{B_0\}$	B_0 is maximal
$B_0 \vee A_1$	$\{B_0\}$	\emptyset	true in I_C
$\neg B_0 \vee A_1$	$\{B_0\}$	$\{A_1\}$	A_1 is maximal
$\neg B_0 \vee B_1 \vee \neg B_0 \vee B_1$	$\{B_0, A_1\}$	$\{B_1\}$	B_1 is maximal
$\neg B_0 \vee A_2 \vee B_1$	$\{B_0, A_1, B_1\}$	\emptyset	true in I_C
$\neg B_0 \vee \neg A_2 \vee B_1$	$\{B_0, A_1, B_1\}$	\emptyset	
$\neg B_1 \vee B_2$	$\{B_0, A_1, B_1\}$	$\{B_2\}$	

The resulting interpretation $I = \{B_0, A_1, B_1, B_2\}$ is a model of all clauses.

Another consequence of Theorem 3.9 is the compactness of closed clausal logic.

3.12. THEOREM. *A set N of ground clauses is unsatisfiable if and only if some finite subset of N is unsatisfiable.*

Ordered resolution with selection

$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \quad \dots \quad C_n \vee A_n \vee \cdots \vee A_n \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

where

- (i) either the subclause $\neg A_1 \vee \cdots \vee \neg A_n$ is selected by S in D , or else $S(D)$ is empty, $n = 1$, and A_1 is maximal with respect to D ,
- (ii) each atom A_i is strictly maximal with respect to C_i , and
- (iii) no clause $C_i \vee A_i \vee \cdots \vee A_i$ contains a selected atom.

Figure 2: (*Standard*) *Ordered Resolution* O_S^\succ

Resolution, as a sound and complete inference rule, provides a suitable deductive basis for refutational theorem proving. For practical purposes the above inference rule is too prolific, though, in that too many clauses can be deduced (the “search space” is too large). We next discuss useful restrictions on resolution that do not impair its completeness.

An inspection of the proof of Theorem 3.8 reveals that we have actually established a stronger result than is stated in the theorem. For instance, resolution is only required on the maximal atom in the side premise, so that corresponding ordering restrictions may be imposed on inferences. Furthermore, it is sufficient to resolve on any of the negative literals in a don’t-care non-deterministic way. We propose selection functions as a corresponding control mechanism. Finally, we will package several inferences into one larger inference step by simultaneously resolving on more than one atom. This has the possible advantage that intermediate results need not be retained.

By a *selection function* we mean a mapping S that assigns to each clause C a (possibly empty) multiset $S(C)$ of negative literals in C . In other words, the function S selects (a possibly empty) negative subclause of C . We say that an atom A , or a literal $\neg A$, is *selected by* C if $\neg A$ occurs in $S(C)$. (There are no selected atoms or literals if $S(C)$ is empty.)

Let \succ be an admissible clause ordering and S be a selection function. The inference system O_S^\succ of ordered resolution with selection is shown in Figure 2. (The subscript and/or superscript in O_S^\succ will be omitted if the relevant information is either clear from the context or intentionally left unspecified. Specific settings of the parameters will be discussed in Sections 6 and 7.) In ordered inferences one resolves either all selected atoms at once or, in case there are no selected atoms, the maximal atom of the main premise. Furthermore, the side premises must contain no selected atoms at all.

A key argument in the proof of Theorem 3.8 is that resolution inferences can be used to reduce certain counterexamples. Ordered resolution is designed so that each resolvent is smaller than the corresponding main premise.

3.13. LEMMA. *If \succ is an admissible ordering and S any selection function, then the conclusion of any inference in O_S^\succ is smaller (with respect to \succ) than the main premise.*

Theorem 3.8 can easily be extended to ordered resolution with selection by applying similar ideas to a model construction that is slightly modified from Definition 3.1. Given \succ , a total admissible ordering on clauses, and a set of clauses N , we again use induction with respect to \succ to define for each clause C (not necessarily in N) a Herbrand interpretation I_C and a set ε_C as follows.

3.14. DEFINITION. Take I_C to be the set $\bigcup_{C \succ D} \varepsilon_D$. Furthermore, if C is a clause that

- (i) is contained in N ;
- (ii) is of the form $C' \vee A$, where A is the maximal literal in C ;
- (iii) is false in I_C ; and
- (iv) nothing is selected in C ;

then $\varepsilon_C = \{A\}$; otherwise, ε_C is the empty set.

Again we say that C *produces* A , and call C a *productive clause*, if $\varepsilon_C = \{A\}$. By the defn[candidate model]*candidate model* for N , denoted by I_N^\succ or simply I_N , we now mean the Herbrand interpretation $\bigcup_{C \in N} \varepsilon_C$ as just defined. As the side-premises of counterexample-reducing inferences are productive clauses, they should not have any selected literals. A related modification of the proof of Theorem 3.8 gives us this refined result:

3.15. THEOREM. *Let N be a set of clauses not containing the empty clause. Let C be the minimal counterexample in N for I_N . Then there exists an inference in O from C such that*

- (i) *its conclusion is a counterexample for I_N and is smaller than C ; and*
- (ii) *C is its main premise and the side premises are productive clauses.*

This reduction property for counterexamples by ordered resolution immediately implies the refutational completeness of the inference system.

3.16. THEOREM. *If N is saturated with respect to O and does not contain the empty clause then I_N is a model of N . The inference system O is therefore refutationally complete.*

This concludes our introduction to saturation-based theorem proving. More sophisticated techniques will be discussed in later chapters. But first we need to outline a rigorous framework for the description of theorem proving strategies.

4. A Framework for Saturation-Based Theorem Proving

All theorem provers employ some deductive inference mechanisms. In the case of refutational provers the goal is to derive a contradiction from any inconsistent set

of input formulas, and the derivation process typically amounts to a saturation of the input set. A straightforward, naive saturation in which one exhaustively applies inferences to previously derived clauses will be hopelessly inefficient in all but the most trivial cases. In a clausal prover each derived clause represents a partial proof or proof attempt, and increases the number of possibilities for constructing a complete proof (or refutation). A (partial) proof (attempt) that is subsumed by another is redundant and should be deleted to avoid useless computations. In most refutational provers, the deductive core accounts for a rather small part of the system, while most of the complexity of the prover derives from the implementation of powerful, yet efficient redundancy elimination and simplification techniques. The degree of sophistication of the latter tools usually distinguishes experimental prototypes from practically useful tools. Unfortunately, comparatively little effort has been devoted to a formal analysis of redundancy and other fundamental concepts of theorem proving strategies, while more emphasis has been placed on investigating the refutational completeness of a variety of modifications of inference rules, such as resolution.

We will next describe a comprehensive framework for modeling the key aspects of theorem proving, such as deduction, deletion, and simplification. In this formalism we will be able to describe a wide range of different theorem proving strategies and to argue about their refutational completeness. The concepts and results in this chapter do not depend on details pertaining to specific syntax or representation of formulas, but apply to *general* clauses containing arbitrary quantifier-free subformulas.

4.1. Theorem Proving Processes

We first develop the minimal prerequisites for a theory of refutational theorem proving with deduction and deletion. We assume that deduction is based on a clausal inference system Γ . The formalization of deletion is more subtle and technically involved in that formulas may only be deleted if one can be certain that this will not prevent the successful completion of the proof search. We will formulate deletion strategies in terms of redundancy criteria for formulas and inferences. Redundancy refers to the states of the theorem proving process, as represented by the collection of clauses that have been derived and retained.

A redundancy criterion is specified by two mappings $\mathcal{R}_{\mathcal{F}}$ and $\mathcal{R}_{\mathcal{I}}$, which associate with each set N of clauses a set of clauses and a set of inferences, respectively, that are deemed to be redundant in the context N .

For example, $\mathcal{R}_{\mathcal{F}}(N)$ will usually contain all tautologies (in the given language), whereas $\mathcal{R}_{\mathcal{I}}(N)$ may contain all inferences the conclusion of which is already an element of N .

4.1. DEFINITION. Let Γ be an inference system. A pair $\mathcal{R} = (\mathcal{R}_{\mathcal{F}}, \mathcal{R}_{\mathcal{I}})$ of mappings from sets of clauses to sets clauses and inferences (by Γ), respectively, is called a *redundancy criterion* if, for all sets of clauses N and N'

Deduction

$$N \triangleright N, M \quad \text{if } M \subseteq C(\Gamma(N))$$

Deletion

$$N, M \triangleright N \quad \text{if } M \subseteq \mathcal{R}(N)$$

Figure 3: *Theorem Proving Derivations* \triangleright

(R1) if $N \subseteq N'$ then $\mathcal{R}_{\mathcal{F}}(N) \subseteq \mathcal{R}_{\mathcal{F}}(N')$ and $\mathcal{R}_{\mathcal{I}}(N) \subseteq \mathcal{R}_{\mathcal{I}}(N')$;

(R2) if $N' \subseteq \mathcal{R}_{\mathcal{F}}(N)$ then $\mathcal{R}_{\mathcal{F}}(N) \subseteq \mathcal{R}_{\mathcal{F}}(N \setminus N')$ and $\mathcal{R}_{\mathcal{I}}(N) \subseteq \mathcal{R}_{\mathcal{I}}(N \setminus N')$; and

(R3) if N is inconsistent, then $N \setminus \mathcal{R}_{\mathcal{F}}(N)$ is also inconsistent.

The criterion is called *effective* (for Γ) if, in addition,

(R4) an inference γ in Γ is in $\mathcal{R}_{\mathcal{I}}(N)$ whenever its conclusion is in $N \cup \mathcal{R}_{\mathcal{F}}(N)$.

The first condition expresses monotonicity of redundancy under the subset relation and, in particular, under the deduction of new clauses. The second condition requires that redundancy be independent of clauses that are redundant in the given context. The third condition states that the removal of redundant clauses preserves inconsistency. Finally, the fourth condition implies that adding its conclusion renders an inference redundant, so that redundancy of inferences can always be achieved by systematic computation of inferences. Inferences in $\mathcal{R}_{\mathcal{I}}(N)$ and clauses in $\mathcal{R}_{\mathcal{F}}(N)$, respectively, are said to be *redundant* (with respect to \mathcal{R} in context N). We emphasize that $\mathcal{R}_{\mathcal{F}}(N)$ need not be a subset of N and that $\mathcal{R}_{\mathcal{I}}(N)$ may contain inferences whose premises are not in N .

A *trivial*, but effective, redundancy criterion for any inference system Γ is given by the mappings $\mathcal{R}_{\mathcal{F}}(N) = \emptyset$ and $\mathcal{R}_{\mathcal{I}}(N) = \{\gamma \in \Gamma \mid C(\gamma) \in N\}$. That is, no clause is redundant in any context, while an inference is taken to be redundant only if its conclusion is already present. More interesting, non-trivial redundancy criteria for resolution-based inference systems will be described later.

At an abstract level a saturation-based theorem prover can be described by a binary relation \triangleright on sets of clauses, called a *transition* or *derivation relation*. We specifically consider derivation relations where each step $N \triangleright N'$ consists of either adding logical consequences (by applying inferences from Γ) or deleting redundant clauses (according to a criterion \mathcal{R} for Γ), cf. Figure 3 (where we use multiset notation for clauses and write, for instance, N, M instead of $N \cup M$). A (finite or countably infinite) sequence $N_0 \triangleright N_1 \triangleright N_2 \triangleright \dots$ is called a (*theorem proving*) *derivation* (based on Γ and \mathcal{R}). The set $N_{\infty} = \bigcup_i \bigcap_{j \geq i} N_j$ of all *persisting clauses* is called the *limit* of the derivation. By a *theorem prover* we mean a procedure that accepts as input a set of clauses N , and produces a derivation $N = N_0 \triangleright N_1 \triangleright N_2 \triangleright \dots$ from N based on a given inference system Γ and redundancy criterion \mathcal{R} . The sets N_i represent the successive states in the theorem proving process; the set N_{∞} its result (which in the case of an infinite derivation is only obtained in the limit).

We consider deductive rules that are consistency-preserving, and condition (R3)

ensures that the deletion of redundant formulas preserves the inconsistency of a set of clauses. It can therefore easily be seen that in a theorem proving derivation either all clause sets N_i are consistent, or all sets are inconsistent.

4.2. LEMMA. *Let $N_0 \triangleright N_1 \triangleright N_2 \triangleright \dots$ be a derivation based on a Γ and \mathcal{R} . Then $\mathcal{R}_{\mathcal{F}}(\bigcup_j N_j) \subseteq \mathcal{R}_{\mathcal{F}}(N_\infty)$ and $\mathcal{R}_{\mathcal{I}}(\bigcup_j N_j) \subseteq \mathcal{R}_{\mathcal{I}}(N_\infty)$. Moreover, the limit set N_∞ is satisfiable if and only if the initial set N_0 is satisfiable.*

PROOF. First note that by the definition of N_∞ , a clause that is in $(\bigcup_j N_j)$ but not in N_∞ , must be in some set $\mathcal{R}_{\mathcal{F}}(N_i)$. Therefore we have $(\bigcup_j N_j) \setminus N_\infty \subseteq \bigcup_j \mathcal{R}_{\mathcal{F}}(N_j)$. Moreover, by condition (R1), $\bigcup_j \mathcal{R}_{\mathcal{F}}(N_j) \subseteq \mathcal{R}_{\mathcal{F}}(\bigcup_j N_j)$. As a consequence, we have $(\bigcup_j N_j) \setminus \mathcal{R}_{\mathcal{F}}(\bigcup_j N_j) \subseteq N_\infty$. Applying condition (R1) again, we may infer that $\mathcal{R}((\bigcup_j N_j) \setminus \mathcal{R}_{\mathcal{F}}(\bigcup_j N_j)) \subseteq \mathcal{R}(N_\infty)$ (where \mathcal{R} may be either $\mathcal{R}_{\mathcal{F}}$ or $\mathcal{R}_{\mathcal{I}}$). Using condition (R2) we obtain $\mathcal{R}(\bigcup_j N_j) \subseteq \mathcal{R}(N_\infty)$.

For the second part, note that by the soundness of the inference system Γ and the compactness of clausal logic, the set N_0 is satisfiable if and only if $\bigcup_j N_j$ is satisfiable. Thus, if N_0 is satisfiable, then N_∞ , as a subset of $\bigcup_j N_j$, is also satisfiable. On the other hand, if N_0 is unsatisfiable, then by condition (R3) the set $(\bigcup_j N_j) \setminus \mathcal{R}_{\mathcal{F}}(\bigcup_j N_j)$ is unsatisfiable as well, and N_∞ is a superset. \square

A refutationally complete theorem prover derives a contradiction from any inconsistent initial set N_0 . Evidently “sufficiently many” inferences must be computed to ensure refutational completeness; a more precise characterization is based on two important concepts, saturation and fairness.

We say that N is *saturated up to redundancy* (with respect to Γ and \mathcal{R}) if all inferences in Γ with non-redundant premises from N are redundant in N , i.e., $\Gamma(N \setminus \mathcal{R}_{\mathcal{F}}(N)) \subseteq \mathcal{R}_{\mathcal{I}}(N)$. For example, a set N is saturated with respect to the trivial redundancy criterion \mathcal{R} if and only if $\mathcal{C}(\Gamma(N)) \subseteq N$. In general, saturation up to redundancy can be achieved by a “fair” computation of inferences from persisting, non-redundant clauses.

A derivation $N_0 \triangleright N_1 \triangleright N_2 \triangleright \dots$ based on an inference system Γ and an *effective* redundancy criterion \mathcal{R} is called *fair* if the conclusion of every non-redundant inference in Γ from non-redundant formulas in N_∞ is either an element of, or redundant in, $\bigcup_j N_j$; that is, if $N' = N_\infty \setminus \mathcal{R}_{\mathcal{F}}(N_\infty)$ then

$$\mathcal{C}(\Gamma(N') \setminus \mathcal{R}_{\mathcal{I}}(N')) \subseteq \bigcup_j N_j \cup \mathcal{R}_{\mathcal{F}}(\bigcup_j N_j).$$

Intuitively fairness means that no inference in Γ from non-redundant persisting formulas be delayed indefinitely. A fair derivation can be constructed by exhaustively applying inferences to persisting formulas.

4.3. THEOREM. *If a derivation, based on an inference system Γ and an effective redundancy criterion \mathcal{R} , is fair then its limit is saturated up to redundancy with respect to Γ and \mathcal{R} .*

PROOF. Let γ be an inference from non-redundant clauses in N_∞ and \mathcal{C} its conclusion. If \mathcal{C} is in $\bigcup_j N_j$ then, by the condition (R4), γ is redundant in $\bigcup_j N_j$ and, by Lemma 4.2, also redundant in N_∞ . On the other hand, if the clause \mathcal{C} is redundant in $\bigcup_j N_j$ and, hence, in N_∞ , then by condition (R4) the inference γ is redundant in N_∞ . We conclude that N_∞ is saturated up to redundancy. \square

Fairness provides an effective way of saturation for effective redundancy criteria.

Propositional inference systems such as O_S^\succ can only be approximately lifted to non-ground clauses. That is, the corresponding non-ground versions of the inference rules usually have more ground instances than needed. Fortunately, the concepts of redundancy and theorem proving derivations are insensitive to such extensions of the inference system. If Γ' is an inference system extending Γ , that is, $\Gamma \subseteq \Gamma'$, then any redundancy criterion \mathcal{R} for Γ can be extended to a redundancy criterion \mathcal{R}' for Γ' by defining $\mathcal{R}'_{\mathcal{F}}(N) = \mathcal{R}_{\mathcal{F}}(N)$ and $\mathcal{R}'_{\mathcal{I}}(N) = \mathcal{R}_{\mathcal{I}}(N) \cup (\Gamma' \setminus \Gamma)$, for all sets of clauses N . With this definition, the additional inferences in $\Gamma' \setminus \Gamma$ become redundant in any context. This *standard extension* of a redundancy criterion is useful if the inferences in $\Gamma' \setminus \Gamma$ are optional for a theorem prover. If \mathcal{R} is effective, so is its standard extension \mathcal{R}' . We say that a *derivation is based on an extension* of Γ and \mathcal{R} whenever there is an inference system $\Gamma' \supseteq \Gamma$ such that the derivation is based on Γ' and the standard extension of \mathcal{R}' . In that case, if the derivation is fair with respect to inferences in Γ the derivation is also fair with respect to inferences in Γ' , and vice versa. The limit of any such derivation is, therefore, saturated with respect to Γ and \mathcal{R} .

4.2. Counterexample-Reducing Inference Systems

Candidate models and reduction of counterexamples are key concepts in establishing the refutational completeness of refutational theorem proving systems. In making these concepts more explicit, we will also arrive at a useful notion of redundancy for resolution. Throughout this section we assume that all expressions are ground and that \succ is an admissible clause ordering. Clauses with variables will be discussed later.

4.2.1. Candidate Models and Counterexamples

Let I be a mapping, called a *model functor*, that assigns to each set N of ground clauses not containing a contradiction an interpretation I_N , called a *candidate model*. If I_N is a model of N , then N is evidently satisfiable. If, on the other hand, some clause \mathcal{C} in N is false in I_N (a *counterexample* for I_N), then N must contain a *minimal* such counterexample with respect to \succ . We say that an inference system Γ has the *reduction property for counterexamples* (with respect to I and \succ) if, for all sets N of clauses and minimal counterexamples \mathcal{C} for I_N in N , there exists an inference in Γ from N with main premise \mathcal{C} , side premises that are true in N , and a conclusion \mathcal{D} that is a smaller counterexample for I_N than \mathcal{C} , i.e., $\mathcal{C} \succ \mathcal{D}$. Inference systems with this property are refutationally complete (with respect to

the trivial redundancy criterion).

4.4. THEOREM. *If Γ has the reduction property for counterexamples and N is saturated with respect to Γ , that is, $C(\Gamma(N)) \subseteq N$, then N is either satisfiable or else contains the empty clause.*

PROOF. Suppose Γ has the reduction property for counterexamples with respect to some model functor I , and N is a set of clauses that does not contain the empty clause. If N contains a counterexample for I_N , it also contains a minimal counterexample \mathcal{C} . By the reduction property, \mathcal{C} can be reduced to an even smaller counterexample \mathcal{D} , which contradicts the minimality of \mathcal{C} . Thus N can not contain a counterexample for I_N , which implies that I_N is a model of N . \square

4.2.2. The Standard Redundancy Criterion

Inference systems with the reduction property for counterexamples admit a non-trivial redundancy criterion, called the *standard redundancy criterion*, that is based on the given clause ordering, but largely independent of the inference system. The intention in defining the criterion is to identify those clauses and inferences which cannot involve any minimal counterexample.

A clause \mathcal{C} is called *redundant* with respect to a set N of clauses if there exist clauses $\mathcal{C}_1, \dots, \mathcal{C}_k$ in N such that $\mathcal{C}_1, \dots, \mathcal{C}_k \models \mathcal{C}$ and $\mathcal{C} \succ \mathcal{C}_i$, for all i with $1 \leq i \leq k$. Note that \mathcal{C} need not be an element of N . For example, tautological clauses $\mathcal{C} \vee A \vee \neg A$ are redundant in any context N . We define a mapping $\mathcal{R}_{\mathcal{F}}^>$ by taking $\mathcal{R}_{\mathcal{F}}^>(N)$ to be the set of all redundant clauses with respect to N . By $N_{\mathcal{C}}$ we denote the set of clauses in N that are smaller than \mathcal{C} . A redundant clause \mathcal{C} logically follows from $N_{\mathcal{C}}$, and therefore can not be a minimal counterexample in N for any interpretation.

If an inference reduces a minimal counterexample (i.e., the main premise), then its conclusion, but none of the side premises, is a counterexample, which suggests the following definition. An inference with main premise \mathcal{C} , side premises $\mathcal{C}_1, \dots, \mathcal{C}_n$, and conclusion \mathcal{D} is called *redundant* (with respect to N), if there exist clauses $\mathcal{D}_1, \dots, \mathcal{D}_k$ in $N_{\mathcal{C}}$ such that $\mathcal{D}_1, \dots, \mathcal{D}_k, \mathcal{C}_1, \dots, \mathcal{C}_n \models \mathcal{D}$. By $\mathcal{R}_{\mathcal{I}}^>(N)$ we denote the set of redundant inferences in Γ with respect to N . We emphasize that $\mathcal{R}_{\mathcal{I}}^>(N)$ will usually contain inferences whose premises are not in N .

Standard redundancy is mainly useful for inference systems with the reduction property, but the notion is well-defined for any inference system and ordering.

4.5. LEMMA. *If $N \subseteq N'$, then $\mathcal{R}_{\mathcal{F}}^>(N) \subseteq \mathcal{R}_{\mathcal{F}}^>(N')$. Furthermore, if a clause \mathcal{C} is redundant in N , then there exist non-redundant clauses $\mathcal{C}_1, \dots, \mathcal{C}_k$ in N , such that $\mathcal{C}_1, \dots, \mathcal{C}_k \models \mathcal{C}$ is valid and $\mathcal{C} \succ \mathcal{C}_1, \dots, \mathcal{C}_k$. Consequently, $\mathcal{R}_{\mathcal{F}}^>(N) \subseteq \mathcal{R}_{\mathcal{F}}^>(N \setminus \mathcal{R}_{\mathcal{F}}^>(N))$, for all sets of clauses N .*

PROOF. The first part follows immediately from the definition of redundancy. For the second part, suppose \mathcal{C} is redundant in N . Let $N' = \mathcal{C}_1, \dots, \mathcal{C}_k$ be a minimal subset of N (with respect to the multiset ordering \succ_{mul}), such that $\mathcal{C}_1, \dots, \mathcal{C}_k \models \mathcal{C}$ and $\mathcal{C} \succ \mathcal{C}_j$, for all j . Then the clauses \mathcal{C}_j are all non-redundant. \square

The lemma indicates that a redundant clause C in N logically follows from $N_C \setminus \mathcal{R}_{\mathcal{F}}^{\succ}(N)$.

4.6. LEMMA. *If $N \subseteq N'$, then $\mathcal{R}_{\mathcal{I}}^{\succ}(N) \subseteq \mathcal{R}_{\mathcal{I}}^{\succ}(N')$. Moreover, $\mathcal{R}_{\mathcal{I}}^{\succ}(N) \subseteq \mathcal{R}_{\mathcal{I}}^{\succ}(N \setminus \mathcal{R}_{\mathcal{F}}^{\succ}(N))$, for all sets of clauses N .*

The proof uses Lemma 4.5.

4.7. THEOREM. *\mathcal{R}^{\succ} is a redundancy criterion.*

PROOF. Lemmas 4.5 and 4.6 indicate that properties (R1) and (R2) are satisfied. In addition, the redundancy criterion \mathcal{R}^{\succ} preserves inconsistency, as required by (R3). \square

An inference with main premise C and conclusion D is called *reductive* with respect to an ordering \succ if $C \succ D$. An inference system is called *reductive* if all its inferences are.

4.8. THEOREM. *The standard redundancy criterion \mathcal{R}^{\succ} is effective for any inference system that is reductive with respect to \succ .*

If an inference system Γ has the reduction property, the subset of its reductive inferences also satisfies the reduction property. Therefore one may ignore non-reductive inferences, and standard redundancy provides an effective criterion.

4.9. THEOREM. *Let Γ be an inference system that satisfies the reduction property with respect to \succ , and let N be a set of clauses that is saturated up to redundancy with respect to Γ and the standard criterion \mathcal{R}^{\succ} . Then N is unsatisfiable if, and only if, it contains a contradiction.*

PROOF. Suppose N is saturated and unsatisfiable, but contains no contradiction, and let M be the set $N \setminus \mathcal{R}_{\mathcal{F}}^{\succ}(N)$. Consider the interpretation I_M . If I_M is not a model of M , then M contains a minimal counterexample C for I_M . Since Γ has the reduction property there is an inference from M with main premise C , side premises C_1, \dots, C_k , and a conclusion D , such that D is a smaller counterexample for I_M than C and the clauses C_i are true in I_M . By saturation, this inference is redundant. Thus, there are clauses D_1, \dots, D_m in N_C , all smaller than C , such that D logically follows from $C_1, \dots, C_k, D_1, \dots, D_m$. According to Lemma 4.5 we may assume that each clause D_j is non-redundant (and, therefore, is in M) and true in I_M (for C is the minimal counterexample for I_M). But this implies that D is true in I_M , which is a contradiction. In sum, I_M is a model of M , and also of N . \square

To summarize, inference systems that satisfy the reduction property for counterexamples are refutationally complete and also compatible with application of the standard redundancy criterion. Standard redundancy, as will be seen below, is a powerful concept that justifies most, if not all, of the common simplification and

Ordered resolution with selection

$$\frac{C_1 \vee A_{i1} \vee \dots \vee A_{ik_1} \quad \dots \quad C_n \vee A_{in} \vee \dots \vee A_{nk_n} \quad \neg A_1 \vee \dots \vee \neg A_n \vee D}{C_1 \sigma \vee \dots \vee C_n \sigma \vee D \sigma}$$

where σ is a most general simultaneous solution of all unification problems $A_{i1} = \dots = A_{ik_i} = A_i$, where $1 \leq i \leq n$, and

- (i) either A_1, \dots, A_n are selected in D , or else nothing is selected in D , $n = 1$, and $A_1 \sigma$ is maximal in $D \sigma$,
- (ii) each atom $A_{ii} \sigma$ is strictly maximal with respect to $C_i \sigma$, and
- (iii) no clause $C_i \vee A_{i1} \vee \dots \vee A_{ik_i}$ contains a selected atom.

Figure 4: *Ordered Resolution for First-Order Standard Clauses* O_S^\succ

deletion techniques used in refutational theorem provers. Next we discuss issues related to lifting these methods to clauses with variables.

4.3. A Simple Resolution Prover for First-Order Clauses

We will use an extended example to illustrate how the theoretical concepts of theorem proving derivations and redundancy can be applied to a realistic model of a resolution-based theorem prover with fundamental simplification techniques. The prover is applicable to *non-ground*, *standard* clauses, so that we also need to address the issue of lifting from the ground level to non-ground clauses.

Figure 4 defines a generalization of ordered resolution to standard first-order (i.e., non-ground) clauses via unification. If \mathcal{O} is a set of clauses and if C is a clause, by $O_S^\succ(\mathcal{O}, C)$ we denote the set of all inferences in O_S^\succ for which one of the premises is the clause C and the other premises are clauses in \mathcal{O} . We assume that \succ is an admissible ordering on ground expressions that has been extended to non-ground expressions (atoms, literals, and clauses) as follows: $E \succ E'$ iff $E\sigma \succ E'\sigma$, for all ground substitutions σ . The extended ordering on non-ground expressions is only a partial (well-founded) ordering, even though the ordering is total on ground atoms. We also implicitly assume that different premises and the conclusion have no variables in common; variables are renamed if necessary. The selection function S is defined both for ground and non-ground clauses: $S(C)$ is a (possibly empty) sequence of negative atoms in C . The inference rule shown in Figure 4 coincides with the earlier inference rule in Figure 2 when all premises are ground. In that case, unifiability of atoms coincides with (syntactic) equality, and, since the ordering on ground expressions is total, the complement of the ordering \succ is \preceq and the complement of \prec is \succeq .

The prover employs specific redundancy criteria. We say that a clause C *subsumes* a clause D (or that D *is subsumed by* C) if and only if there exists a substitution

Tautology deletion

$$\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O} \Rightarrow \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \quad \text{if } C \text{ is a tautology}$$

Forward subsumption

$$\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O} \Rightarrow \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \quad \text{if some clause in } \mathcal{P} \cup \mathcal{O} \text{ subsumes } C$$

Backward subsumption

$$\mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O} \Rightarrow \mathcal{N} \mid \mathcal{P} \mid \mathcal{O}$$

$$\mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C\} \Rightarrow \mathcal{N} \mid \mathcal{P} \mid \mathcal{O}$$

if some clause in \mathcal{N} properly subsumes C

Forward reduction

$$\mathcal{N} \cup \{C \vee L\} \mid \mathcal{P} \mid \mathcal{O} \Rightarrow \mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O}$$

if there is a clause $D \vee L'$ in $\mathcal{P} \cup \mathcal{O}$ such that $\bar{L} = L'\sigma$ and $D\sigma \subseteq C$

Backward reduction

$$\mathcal{N} \mid \mathcal{P} \cup \{C \vee L\} \mid \mathcal{O} \Rightarrow \mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O}$$

$$\mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C \vee L\} \Rightarrow \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C\}$$

if there is a clause $D \vee L'$ in \mathcal{N} such that $\bar{L} = L'\sigma$ and $D\sigma \subseteq C$

Clause processing

$$\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O} \Rightarrow \mathcal{N} \mid \mathcal{P} \cup \{C\} \mid \mathcal{O}$$

Inference computation

$$\emptyset \mid \mathcal{P} \cup \{C\} \mid \mathcal{O} \Rightarrow \mathcal{N} \mid \mathcal{P} \mid \mathcal{O} \cup \{C\}$$

where $\mathcal{N} = C(\mathcal{O}_S^>(\mathcal{O}, C))$

Figure 5: *The Resolution Prover RP*

σ such that $C\sigma$ is a sub-multiset of D . If C subsumes D , but not vice-versa, then C is said to *properly subsume* D . Subsumption defines a well-founded ordering on clauses. Two clauses C and D are said to be *variants* of each other if they mutually subsume each other.

The following resolution rule is of interest as it enables a subsequent subsumption:

Subsumption resolution

$$\frac{D \vee L \quad C \vee D\sigma \vee \bar{L}\sigma}{C \vee D\sigma}$$

This inference is used in combination with deletion, as the conclusion of this inference renders the second premise redundant. The inference *reduces* $C \vee D\sigma \vee \bar{L}\sigma$ to $C \vee D\sigma$.

Figure 5 depicts a binary relation \Rightarrow that formalizes a resolution prover RP with tautology elimination, subsumption and subsumption resolution. The prover operates on triples $(\mathcal{N} \mid \mathcal{P} \mid \mathcal{O})$ of clause sets \mathcal{N} , \mathcal{P} , and \mathcal{O} that represent a *state* of the theorem proving process in terms of newly derived resolvents, “processed” clauses, and “old” clauses. Initial states are of the form $(\mathcal{N} \mid \emptyset \mid \emptyset)$, where \mathcal{N} is a

finite set of possibly non-ground standard clauses.

The first five rules deal with redundancy elimination and simplification. Newly derived resolvents may be deleted if they are tautologies or are subsumed by processed or old clauses. In addition they may be simplified by reduction with old clauses. *Forward subsumption* allows us to remove a newly derived clause whenever it is subsumed by a processed or old clause, but *backward subsumption* is based on proper subsumption. A clause may be moved from the “new” set to the “processed” set at any time, but preferably after simplification. Once all new clauses have been processed, new resolution inferences are computed between some selected processed clause C and all old clauses, after which C becomes an old clause itself. The distinction between “processed” and “old” is useful for achieving fairness in inference computation.

Next we show that derivations by \Rightarrow represent theorem proving derivations on the sets of ground instances represented by the successive states. We denote by $G(C)$ the set of all *ground instances* $C\sigma$ of a clause C . G is extended to sets of clauses by taking the union of the sets of ground instances of the clauses in the set. If $\mathcal{S}_i = (\mathcal{N}_i \mid \mathcal{P}_i \mid \mathcal{O}_i)$ is a state, we sometimes, ambiguously, identify \mathcal{S}_i with the set $\mathcal{N}_i \cup \mathcal{P}_i \cup \mathcal{O}_i$. In particular, by $G(\mathcal{S}_i)$ we denote the set $G(\mathcal{N}_i) \cup G(\mathcal{P}_i) \cup G(\mathcal{O}_i)$ of all ground instances of clauses present in state \mathcal{S}_i .

Consider a derivation

$$\mathcal{N}_0 \mid \emptyset \mid \emptyset \Rightarrow \mathcal{N}_1 \mid \mathcal{P}_1 \mid \mathcal{O}_1 \Rightarrow \mathcal{N}_2 \mid \mathcal{P}_2 \mid \mathcal{O}_2 \Rightarrow \dots$$

on states $\mathcal{S}_i = (\mathcal{N}_i \mid \mathcal{P}_i \mid \mathcal{O}_i)$, where \mathcal{N}_0 is a finite set of clauses and $\mathcal{P}_0 = \mathcal{O}_0 = \emptyset$. Let N_i be an abbreviation for the sets of ground instances $G(\mathcal{S}_i)$. We will first show that the sequence N_0, N_1, N_2, \dots represents a theorem proving derivation \triangleright based on some extension (in the sense of Section 4.1) of ordered resolution O_S^\triangleright with standard redundancy $\mathcal{R}^\triangleright$. The inferences on which the derivations in RP are based include those in O_S^\triangleright . But there are also other inferences applied, such as subsumption resolution. Moreover, even if \mathcal{S}_{i+1} results from \mathcal{S}_i by a step of inference computation in (the non-ground version of) O_S^\triangleright , the clauses in $N_{i+1} \setminus N_i$ might not all be representable as conclusions of inferences in (the ground version) of O_S^\triangleright . For instance, the inference

$$\frac{p(f(x, a)) \quad \neg p(f(y, z)) \vee \neg p(f(z, y))}{\neg p(f(a, x))}$$

will be in O_S^\triangleright for many orderings, if nothing is selected in the second premise. Its ground instance

$$\frac{p(f(b, a)) \quad \neg p(f(b, a)) \vee \neg p(f(a, b))}{\neg p(f(a, b))}$$

however, is not in O_S^\triangleright , if either $p(f(a, b)) \triangleright p(f(b, a))$, or else if $\neg p(f(b, a))$, but not $\neg p(f(a, b))$, is selected. Selection is generally not compatible with instantiation.

4.10. LEMMA. *If $\mathcal{S} \Rightarrow \mathcal{S}'$ then $G(\mathcal{S}) \triangleright^* G(\mathcal{S}')$, with \triangleright based on some extension of O_S^\triangleright and \mathcal{R} .*

PROOF. We proceed by a case analysis over the definition of \Rightarrow .

Tautology elimination: Suppose $S = (\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O})$ and $S' = (\mathcal{N} \mid \mathcal{P} \mid \mathcal{O})$, where C is a tautology. Then any instance of C is also a tautology, and hence redundant. In other words, $G(S) \setminus G(S') \subseteq \mathcal{R}_{\mathcal{F}}(G(S))$.

Forward subsumption: Let $S = (\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O})$ and $S' = (\mathcal{N} \mid \mathcal{P} \mid \mathcal{O})$ with C subsumed by some D in $\mathcal{P} \cup \mathcal{O}$, that is, $D\sigma \subseteq C$, for some substitution σ . If $D\sigma = C$, then $G(C) \subseteq G(D)$, hence $G(S) = G(S')$. Otherwise, $D\sigma \subset C$ and $G(C) \subseteq \mathcal{R}_{\mathcal{F}}(\{G(D)\})$. In both cases, $G(S) \triangleright G(S')$.

Backward subsumption: similar.

Forward reduction: Let $S = (\mathcal{N} \cup \{C \vee L\} \mid \mathcal{P} \mid \mathcal{O})$ and $S' = (\mathcal{N} \cup \{C\} \mid \mathcal{P} \mid \mathcal{O})$, with $D \vee L'$ in $\mathcal{P} \cup \mathcal{O}$ such that $\bar{L} = L'\sigma$ and $D\sigma \subseteq C$. Reduction can be viewed as a two-step process in which first C is derived by subsumption resolution and then $C \vee L$ is deleted by subsumption. Let $S'' = (\mathcal{N} \cup \{C \vee L, C\} \mid \mathcal{P} \mid \mathcal{O})$. Clearly, $D \vee L', C \vee L \models C$, and therefore $G(S) \triangleright G(S'')$ by a step of deduction in an extended inference system. Moreover, $C \vee L$ is properly subsumed by C so that $G(C \vee L) \subseteq \mathcal{R}_{\mathcal{F}}(G(S''))$, and $G(S'') \triangleright G(S')$ by a step of deletion. Altogether, $G(S) \triangleright^* G(S')$.

Backward reduction: similar.

Clause processing: Here the set of ground clauses represented by the proof state is not changed so that $G(S) \triangleright G(S')$ is trivially true.

Inference selection: Inferences in $\mathcal{O}_S^\triangleright$ produces logical consequences of clauses in S , so that the ground instances of the new clauses in \mathcal{N} can be viewed as derived in an appropriately extended inference system. Therefore $G(S) \triangleright G(S')$. \square

Since the preceding lemma does not reveal much information about the inference system that is the basis of the derivations on the ground level, the real work of showing refutational completeness of RP will be concerned with issues of fairness. More precisely, we have to identify criteria on the level of proof states that will ensure fairness of the corresponding ground derivations N_0, N_1, \dots with respect to $\mathcal{O}_S^\triangleright$, and standard redundancy \mathcal{R} , for certain selection functions S' derived from S .

Let S_∞ be the triple $(\mathcal{N}_\infty \mid \mathcal{P}_\infty \mid \mathcal{O}_\infty)$. A sequence of states S_i is called *fair* if $\mathcal{N}_\infty = \mathcal{P}_\infty = \emptyset$. That is, we require that each clause will be eventually processed and that each processed clause will eventually be selected for inference computation.⁴

4.11. LEMMA. *For any fair sequence of states $S_1 \Rightarrow S_2 \Rightarrow \dots$ we have $G(S_\infty) \supseteq N_\infty \setminus \mathcal{R}(N_\infty)$, where N_∞ denotes the limit of the ground derivation $N_i = G(S_i)$ represented by the sequence of states.*

PROOF. Assume that C is a ground clause in $N_\infty \setminus \mathcal{R}(N_\infty)$, in particular, $C \in N_j$, for all $j \geq i$, with some $i \geq 0$. Being non-redundant, C is not a tautology. If D is

⁴Fairness can be ensured, for instance, by selecting clauses of minimal index for inference computation, where index refers to a weight function that is monotonically increasing both in the size of a clause and the number of the state in which it was produced. The latter component is needed to avoid potentially unfair selection strategies when some clause is generated infinitely often and a larger clause is never selected for inference computation.

a clause in some of the \mathcal{N}_j , $j \geq i$, such that C is an instance of D , then, as the sequence is fair, D will be deleted from \mathcal{N}_{l-1} at some subsequent step $l-1 \geq j$. From Lemma 4.10 and from the definition of RP this can only be the case if C is also instance of a clause D' in some $\mathcal{P}_l \cup \mathcal{O}_l$. (Since C is not redundant, D cannot be deleted from \mathcal{N} by forward reduction as the reduct that would be added would properly subsume D . Therefore, either D is subsumed by some D' in $\mathcal{P}_l \cup \mathcal{O}_l$, or else D is moved from \mathcal{N}_{l-1} to \mathcal{P}_l by a step of clause processing.) As \mathcal{P}_∞ is empty, if D' is in \mathcal{P}_l , it will eventually also be in $\mathcal{O}_{l'}$, for some $l' > l$. Then, D' will be in each of the \mathcal{O}_k , with $k \geq l'$. (Again we make use of the fact that since C is not redundant, D' cannot be deleted by backward reduction or backward subsumption.) Thus, we have shown that there is a clause D' in \mathcal{S}_∞ having C as its ground instance. \square

It is a classical result that (unrestricted) resolution inferences can be lifted in that any inference from ground instances of clauses can be obtained as instances of inferences from the clauses themselves. In our setting a particular technical problem arises in that the selection function S need not be compatible with substitution. If a ground clause is an instance of two different non-ground clauses, which of the two possibly different selections should be inherited by the instance? Fortunately, these ambiguities are not critical. Suppose we have a selection function S and a set M of clauses with ground instances $K = G(M)$. Then let S_M denote an arbitrary new selection function for which (i) if C is in K , then $S_M(C) = S(D)\sigma$, for some D in M such that $C = D\sigma$; and (ii) $S_M(C) = S(C)$, if C is not in K . In other words, for any clause C in K , $S_M(C)$ needs to coincide with $S(D)$ for at least one clause D in M that has C as one of its ground instances. Depending on the choice of D for C , there may be different such functions S_M . Any choice of S_M gives us the required degree of compatibility between selection on the ground and non-ground level, respectively, as formalized by this lifting lemma:

4.12. LEMMA (Lifting Lemma). *Let M be a set of clauses and $K = G(M)$. If*

$$\frac{C_1 \quad \dots C_n \quad C_0}{C}$$

is an inference in $\mathcal{O}_{S_M}^\times(K)$ then there exist clauses C'_i in M , a clause C' , and a ground substitution σ such that

$$\frac{C'_1 \quad \dots C'_n \quad C'_0}{C'}$$

is an inference in $\mathcal{O}_S^\times(M)$, $C_i = C'_i\sigma$, and $C = C'\sigma$.

PROOF. We choose for the premises those clauses C'_i in M which were used for defining $S_M(C_i)$ as the multiset of literals corresponding to the multiset selected by $S(C'_i)$. In that case a resolution inference from the C'_i exists and satisfies the restrictions about \succ and S . \square

4.13. THEOREM. *If $S_0 \Rightarrow S_1 \Rightarrow \dots$ is a fair derivation, then S_0 is unsatisfiable if and only if S_∞ contains the empty clause.*

PROOF. Let again N_i denote $G(S_i)$. From Lemma 4.10 we may infer that the sequence of sets N_i is a theorem proving derivation for which, by Lemma 4.11, we have $N_\infty \setminus \mathcal{R}(N_\infty) \subseteq \mathcal{O}_\infty$. We will show that N_∞ is saturated with respect to $\mathcal{O}_{S_\infty}^\gamma$, where S_∞ is an arbitrary selection function derived from S and \mathcal{O}_∞ . Let γ be an inference in $\mathcal{O}_{S_\infty}^\gamma$ from non-redundant premises in $G(\mathcal{O}_\infty)$ having conclusion C . By the lifting lemma there is an inference in \mathcal{O}_S^γ from clauses C'_i in \mathcal{O}_∞ , $1 \leq i \leq n$, such that the conclusion C' has C as a ground instance. All inferences by \mathcal{O}_S^γ from the clauses C'_i will have been considered when the “youngest” clause C'_i becomes old. Therefore γ is redundant in some $G(S_j)$. By Lemma 4.10, γ is also redundant in $G(\mathcal{O}_\infty) = G(S_\infty)$. Therefore, N_∞ is saturated up to redundancy with respect to $\mathcal{O}_{S_\infty}^\gamma$ and \mathcal{R} , and we may infer that either N_0 is satisfiable or else N_∞ (and hence S_∞) contains the empty clause. S_0 is satisfiable if and only if N_0 is satisfiable, from which the assertion follows. \square

Our investigation of the properties of RP shows that there are considerable technical complications in proving that refutational completeness is preserved by forward subsumption and strict backward subsumption. But if one admits a more liberal notion of backward subsumption, a ground instance $C = D\sigma = D'\sigma'$ may persist, even though neither D nor D' persists on the non-ground level. Thus more complex criteria have to be employed and checked by the prover to ensure fairness.

5. General Resolution

Ordered resolution with selection is a versatile inference system that can be adapted to specific applications not only by adjusting the two parameters of ordering and selection function, but also by exploiting redundancy criteria. These advantages are especially useful for the inference rules on general clauses, a generalization we proceed to describe next. General clauses are multisets of arbitrary quantifier-free formulae, denoting the disjunction of their elements. In this and the subsequent sections all expressions are assumed to be ground, *unless indicated otherwise*. Lifting the new inference systems to non-ground clauses can be done in a way similar to what we have described for ordered resolution on standard clauses in the context of the prover RP in Section 4.3. In Section 9 we will briefly describe refined lifting methods involving, for instance, constrained clauses.

5.1. Selection Functions

First we need to generalize the notion of a selection function. Intuitively, if \mathcal{C} is the minimal counterexample to a candidate model I then one of the sequences selected by $S(\mathcal{C})$ should be true in I , so that \mathcal{C} can be reduced to a smaller counterexample by resolving on selected atoms only.

A (general) *selection function* is a mapping S that assigns to each general clause C a (possibly empty) set $S(C)$ of nonempty sequences of (distinct) atoms in C such that either $S(C)$ is empty or else, for all interpretations I in which C is false, there exists a sequence A_1, \dots, A_k in $S(C)$, all atoms of which are true in I . A sequence A_1, \dots, A_k in $S(C)$ is said to be *selected* (by S). Sometimes the atoms A_i are also called *selected*, especially if a sequence in $S(C)$ consists of a single atom. If $S(C)$ is empty, the clause C contains no selected atom.

For example, possible choices for selection in the clause $C = (\neg A \wedge \neg B, \neg A', B')$ are $S_1(C) = \{(A, A'), (B, A')\}$, $S_2(C) = \{A, B\}$, and $S_3(C) = \{A'\}$.

Verifying that a set $S(C)$ satisfies the required conditions for given S and C is a computationally hard problem in general, though sufficient (syntactic) criteria can be specified in special cases. For instance, if C is a standard clause, then those atoms that occur in negative literals of C are precisely the ones that may be selected, and any set of sequences of negative atoms forms a legal selection. For general clauses, selection can be based on a suitable notion of the *polarity* of a subformula in an expression. We say that a subformula of F' in $E[F']$ is *positive* (respectively, *negative*), if $E[F'/\top]$ (respectively, $E[F'/\perp]$) is a tautology. Thus, if F' is positive (respectively, negative) in E , then F' (respectively, $\neg F'$) logically implies E .

For example, in a disjunction $A \vee B$ both A and B are positive, whereas in a conjunction $A \wedge B$ the two subformulas A and B are neither positive nor negative. A subformula may occur both positively and negatively (e.g., A in $A \vee \neg A$ or $A \equiv A$), in which case the formula is a tautology. For standard clauses $\neg A_1 \vee \dots \vee \neg A_m \vee B_1 \vee \dots \vee B_n$ we obtain the usual notion of polarity in that all atoms A_i are negative and all atoms B_j are positive. It is safe to select any sequence of negative atoms in a general clause, as a negative atom cannot be false in any interpretation in which the clause is false.

Determining whether an atom A is positive or negative in a formula E requires one to check whether $E[A/\top]$ or $E[A/\perp]$ is a tautology, which is again a computationally hard problem. But syntactic criteria allow one to identify certain positive and negative occurrences of atoms in a clause in linear time (in the size of the given clause).

5.1.1. PROPOSITION. (i) F is a positive subformula of F .

(ii) If $\neg G$ is a positive (respectively, negative) subformula of F , then G is a negative (respectively, positive) subformula of F .

(iii) If $G \vee H$ is a positive subformula of F , then G and H are both positive subformulas of F .

(iv) If $G \wedge H$ is a negative subformula of F , then G and H are both negative subformulas of F .

(v) If $G \supset H$ is a positive subformula of F , then G is a negative subformula and H is a positive subformula of F .

(vi) If $G \supset \perp$ is a negative subformula of F , then G is a positive subformula of F .

(vii) F is positive in a clause C if it is an element of C .

5.2. General Ordered Resolution

Let \succ be an ordering and S be a (general) selection function. The inference system O_S^\succ of *general ordered resolution* is depicted in Figure 6. In the case of self-resolution, there is only a main premise, no side premises. In ordered resolution, the last premise is the main premise, while the other premises are the side premises. We occasionally omit the subscript and/or superscript in O_S^\succ if the ordering \succ or the selection function S are clear from the context. In an ordered resolution inference either all selected atoms or, if there are no selected atoms, the maximal atom in the main premise are resolved, and the side premises must not contain any selected atoms.

Note that the order among the two premises is significant. For example, resolution on A in $A \vee B$ and $A \supset C$ yields the resolvent $(\perp \vee B), (\top \supset C)$, which is logically equivalent to B, C . If we exchange the premises, we obtain $(\perp \supset C), (\top \vee B)$, a tautology.

Let us consider two examples.

- | | | |
|-----|--|-----------------------------------|
| (1) | $A \equiv B$ | [input] |
| (2) | $\neg A \vee \neg B$ | [input] |
| (3) | $A \vee B$ | [input] |
| (4) | $(\perp \vee B), (\top \equiv B)$ | [resolving on A in (3) and (1)] |
| (5) | $(\perp \equiv B), (\neg \top \vee \neg B)$ | [resolving on A in (1) and (2)] |
| (6) | $(\perp \vee \perp), (\top \equiv \perp), (\perp \equiv \top), (\neg \top \vee \neg \top)$ | [resolving on B in (4) and (5)] |

Clause (6) is a contradiction as each disjunct simplifies to \perp . Hence, by soundness there is no interpretation in which all three input formulas are true.

A different approach is to repeatedly apply self-resolution to the conjunction of the given (finitely many) input formulas until all atoms have been eliminated:

- | | | |
|-----|--|---------------------------|
| (1) | $(A \equiv B) \wedge (\neg A \vee \neg B) \wedge (A \vee B)$ | [input] |
| (2) | $[(\perp \equiv B) \wedge (\neg \perp \vee \neg B) \wedge (\perp \vee B)]$ | |
| | , $[(\top \equiv B) \wedge (\neg \top \vee \neg B) \wedge (\top \vee B)]$ | [resolving on A in (1)] |
| (3) | $[(\perp \equiv \perp) \wedge (\neg \perp \vee \neg \perp) \wedge (\perp \vee \perp)]$ | |
| | , $[(\top \equiv \perp) \wedge (\neg \top \vee \neg \perp) \wedge (\top \vee \perp)]$ | |
| | , $[(\perp \equiv \top) \wedge (\neg \perp \vee \neg \top) \wedge (\perp \vee \top)]$ | |
| | , $[(\top \equiv \top) \wedge (\neg \top \vee \neg \top) \wedge (\top \vee \top)]$ | [resolving on B in (2)] |

(3) is a contradiction as each disjunct contains a false conjunct.

The examples illustrate two extreme cases in the wide spectrum of possible resolution strategies—ranging from local strategies where replacement of (atomic) sub-formulas is confined to single standard clauses, to global ones in which the entire state of the theorem proving process is modified non-locally at each step.

General ordered resolution with selection

$$\frac{C_1(A_1) \dots C_n(A_n) \quad \mathcal{D}(A_1, \dots, A_n)}{C_1(\perp), \dots, C_n(\perp), \mathcal{D}(\top, \dots, \top)}$$

where (i) either A_1, \dots, A_n is selected by S in \mathcal{D} , or else $S(\mathcal{D})$ is empty, $n = 1$, and A_1 is maximal in \mathcal{D} , (ii) each atom A_i is maximal in C_i , and (iii) no clause C_i contains a selected atom.

Ordered self-resolution

$$\frac{\mathcal{D}(A)}{\mathcal{D}(\perp), \mathcal{D}(\top)}$$

where (i) the atom A is maximal in the premise, and (ii) the premise contains no selected atom.

Figure 6: *General Ordered Resolution* O_S^\succ *5.3. Refutational Completeness*

We establish the refutational completeness of general ordered resolution by showing that the calculus has the reduction property for counterexamples. We essentially use the model functor I from Definition 3.14, but applied to general clauses and with an additional restriction on productive clauses.

Let N be a set of general clauses and \succ be an admissible ordering. We use induction on \succ to define, for each clause C , a Herbrand interpretation I_C and a set ε_C as follows.

5.2. DEFINITION. Take I_C to be the set $\bigcup_{C \succ \mathcal{D}} \varepsilon_{\mathcal{D}}$. If A is the maximal atomic formula of a clause C in N , then $\varepsilon_C = \{A\}$ if (i) $A \notin I_C$, (ii) C is false in I_C , but true in $I_C \cup \{A\}$, and (iii) C contains no selected atom; otherwise, ε_C is the empty set.

We say that C *produces* A , and call C *productive*, if $\varepsilon_C = \{A\}$. Finally, I_N is defined as the Herbrand interpretation $\bigcup_{C \in N} \varepsilon_C$. Whenever we wish to emphasize the dependency on the ordering we write I_N^\succ . The additional restriction on productive clauses compared to Definition 3.14 is that an atom A is produced only if the clause becomes true in the extended interpretation. Hence, a clause such as $A \wedge \perp$ cannot be productive, and, typically, self-resolution (on A) has to be applied to split the clause into two sub-cases.

The inferences in O_S^\succ are reductive in the following sense.

5.3. LEMMA. *Let \succ be an admissible clause ordering and S be any selection function. Then the conclusion of any inference in O_S^\succ is smaller than the main premise.*

Note that the lemma would not hold for resolution of selected non-maximal atoms if conclusions were written as disjunctions, instead of as multisets.

5.4. THEOREM. *Let \succ be an admissible clause ordering, N be a set of clauses not containing a contradiction, and C be a minimal counterexample in N for I_N . Then there exists an inference in O_S^\succ with main premise C , the conclusion of which is a smaller counterexample for I_N than C , and the side premises of which, in the case of ordered resolution, are productive clauses.*

PROOF. Let I be an abbreviation for I_N and \mathcal{D} be the minimal counterexample in N for I . The clause \mathcal{D} is not a contradiction and cannot be productive (as it is false in I). We distinguish two cases.

(i) If \mathcal{D} contains no selected atoms, let A be the maximal atom in \mathcal{D} (with respect to \succ).

(i.1) Suppose A is false in I . Then there is a self-resolution inference with premise \mathcal{D} and resolvent $\mathcal{D}' = \mathcal{D}(\perp), \mathcal{D}(\top)$, where clearly $\mathcal{D} \succ \mathcal{D}'$. Moreover, $\mathcal{D}(\top)$ is false in I , for otherwise \mathcal{D} would produce A . Since A is false in I , the clause $\mathcal{D}(\perp)$ has the same truth value as \mathcal{D} in I , i.e., is false. In short, the clause \mathcal{D}' is a smaller counterexample for I than \mathcal{D} .

(i.2) If A is true in I , it must be produced by some clause C and the clause $C(\perp)$ is therefore false in I . Since $\mathcal{D}(\top)$ is also false in I , the non-clausal resolvent $\mathcal{D}' = C(\perp), \mathcal{D}(\top)$ of C and \mathcal{D} is a smaller counterexample for I than \mathcal{D} .

(ii) If $S(\mathcal{D})$ is nonempty, then by the properties of a selection function it must contain a sequence of atoms A_1, \dots, A_n , all of which are true in I (for \mathcal{D} is false in I). Each atom A_i is produced by some clause C_i in N . Using these clauses as side premises and \mathcal{D} as main premise, we obtain an ordered resolution inference with resolvent

$$\mathcal{D}' = C_1(\perp), \dots, C_n(\perp), \mathcal{D}(\top, \dots, \top).$$

It can easily be shown that \mathcal{D}' is a smaller counterexample for I than \mathcal{D} . □

The theorem indicates that general ordered resolution with selection has the reduction property for counterexamples, and hence is refutationally complete.

5.5. THEOREM (Refutational completeness). *Let \succ be an admissible ordering and S be a selection function. If N is saturated up to standard redundancy under O_S^\succ , then N is unsatisfiable if and only if it contains a contradiction.*

PROOF. If N contains a contradiction, then it is unsatisfiable. If N contains no contradiction, we may use Theorem 4.9 in combination with Theorem 5.4, to infer that it has a model. □

5.4. Applications of Standard Redundancy

The above completeness result, Theorem 5.5, covers a wide range of resolution-based theorem proving strategies. We next show how further refinements may be

obtained by application of standard redundancy. From now on “redundancy” will refer to “standard redundancy,” unless stated otherwise. The following observation will be useful for reasoning about redundancy.

5.6. PROPOSITION. *An inference by ordered resolution with side premises C_1, \dots, C_n , main premise C , resolved atoms A_1, \dots, A_n , and conclusion C' is redundant in N if there exist clauses D_1, \dots, D_k in N such that $C \succ D_1, \dots, D_k$ and*

$$D_1, \dots, D_k, C_1, \dots, C_n, A_1, \dots, A_n \models C'.$$

PROOF. We show that under the given assumptions the clause C' is a logical consequence of $D_1, \dots, D_k, C_1, \dots, C_n$. Suppose all clauses D_i and C_j are true in an interpretation I . Since C' is of the form $C_1[A_1/\perp], \dots, C_n[A_n/\perp], D'$, it is true in I if at least one of the atoms A_i is false in I . On the other hand, if all atoms A_i are true in I , then by the assumptions, C' is also true in I . \square

5.4.1. Polarity-Based Restrictions

An inference is trivially redundant if its conclusion is a tautology. Some such redundant inferences can be detected by an analysis of the polarity of resolved atoms.

5.7. PROPOSITION. *An inference by ordered resolution is redundant if (i) the main premise contains a positive occurrence of some resolved atom A_i , or (ii) some side premise C_i contains a negative occurrence of A_i , or (iii) some side premise C_i contains a positive occurrence of a resolved atom A_j , where $A_j \neq A_i$.*

PROOF. Let $C_1(A_1), \dots, C_n(A_n)$ be the side premises, and $D(A_1, \dots, A_n)$ the main premise of an ordered resolution inference that resolves A_1, \dots, A_n . The resolvent is of the form $D' = C_1(\perp), \dots, C_n(\perp), D(\top, \dots, \top)$. If an atom A_i occurs positively in D then $D(\top, \dots, \top)$, and hence D' , is a tautology. If an atom A_i occurs negatively in $C_i(A_i)$, then $C_i(\perp)$, and hence D' , is a tautology. If an atom A_j , with $A_j \neq A_i$, occurs positively in C_i , then $C_i[A_j/\top]$ is a tautology, as is $C_i[A_i/\perp, A_j/\top]$. Therefore A_j entails $C_i[A_i/\perp]$ and hence the conclusion D' . By Proposition 5.6, the inference is again redundant. \square

In a similar way one can prove:

5.8. PROPOSITION. *An inference by self-resolution is redundant if the resolved atom occurs positively or negatively in the premise.*

Propositions 5.7 and 5.8 suggest additional (polarity) constraints that may be safely attached to inferences in O_S^\sim .

Let us briefly remark that Manna and Waldinger [1980] and Murray [1982] proposed a different restriction for general resolution, where the resolved atom A is required to occur positively in the positive premise and negatively in the negative premise. This requires a different notion of polarity, according to which each subformula is positive or negative (or both). For instance, A and B are considered to

be positive in $A \wedge B$. These polarity constraints are not compatible with ordering constraints or selection and the combination yields an incomplete calculus. For example, take the formulas $A \wedge B$ and $\neg B$ and suppose $A \succ B$ and A and B are considered positive in $A \wedge B$. The inference

$$\frac{A \wedge B \quad \neg B}{(A \wedge \perp), \neg \top}$$

satisfies the polarity constraint: B is positive in the first and negative in the second premise; but not the ordering constraint: B is not maximal in the first premise. (The resolvent, by the way, is a contradictory formula, but not a contradiction in our sense.) On the other hand, the inference

$$\frac{A \wedge B \quad A \wedge B}{(\perp \wedge B), (\top \wedge B)}$$

satisfies the ordering constraint, but not the polarity constraint, as A occurs only positively in both premises. The resolvent of this inference is equivalent to B ; and another resolution step with $\neg B$ yields a contradiction. There is no resolution inference from $A \wedge B$ and $\neg B$ that satisfies both polarity and ordering constraints. In other words, the simultaneous application of both kinds of constraints renders non-clausal resolution incomplete.

Theorem 5.7 already holds for a weaker form of negative polarity in which an atom A is considered negative in $C(A)$ if C implies $C(\perp)$.

5.4.2. Positive Resolution

A general clause is called *positive*, if it is false in the *empty Herbrand interpretation* I_{\perp} (in which all atoms are false). In a positive clause no atoms can be selected, so that $S(C)$ must be the empty set, for any selection function S . Conversely, if a clause is not positive, simply selecting all its non-positive atoms yields a legal selection function, for if a non-positive clause C is false in an interpretation I then I must contain an atom A that is non-positive in C . Also note that a positive clause cannot contain a negative occurrence of an atom, though the converse is not true in general.

Let S be a selection function such that $S(C)$ is the empty set, if C is positive, and $S(C)$ is the set of all sequences (of length one consisting) of a single non-positive atom in C , otherwise. General ordered resolution with such a selection function specializes to positive resolution as shown in Figure 7. The polarity constraints imposed on the inferences derive their justification from Propositions 5.7 and 5.8. As an immediate consequence of these propositions and Theorem 5.5 we get:

5.9. THEOREM. *If N is saturated up to redundancy under GP^{\succ} , then N is unsatisfiable if and only if it contains a contradiction.*

In the case of standard clauses, the main premise of a positive resolution inference must be a non-positive clause. In the general case this restriction is too strong. For

General positive ordered resolution

$$\frac{C(A) \quad D(A)}{C(\perp), D(\top)}$$

where (i) the atom A is maximal in C , (ii) the clause C is positive, and (iii) the atom A is non-positive in D .

General positive ordered self-resolution

$$\frac{D(A)}{D(\perp), D(\top)}$$

where (i) the atom A is maximal in D , and (ii) the clause D is positive.

Figure 7: *General positive resolution* GP^\succ

example, there is no such inference from an inconsistent set consisting of the three atoms A , B , and C , and the positive clause $(\neg A \wedge C, \neg B \wedge C)$, with A the maximal atom.

5.4.3. Partial Replacement Strategies

Resolution inferences essentially represent a case analysis on the truth values of certain atoms. We have formulated the general inferences in such a way that all occurrences of these atoms are replaced simultaneously. The following propositions show that a more selective, “partial” replacement is also possible.

5.10. PROPOSITION. *Let*

$$\frac{C_1(A_1) \quad \dots \quad C_n(A_n) \quad D(A_1, \dots, A_n)}{C_1(\perp), \dots, C_n(\perp), D(\top, \dots, \top)}$$

be a general ordered resolution inference, $\{i_1, \dots, i_k\}$ be any non-empty subset of $\{1, \dots, n\}$, and P be any non-empty set of occurrences of the atoms A_{i_j} in D . If the “partial conclusion”

$$C = C_{i_1}(\perp), \dots, C_{i_k}(\perp), D[\top]_P$$

logically follows from C_1, \dots, C_n and clauses in N smaller than D , then the above inference is redundant in N .

PROOF. Let $\Gamma_1, \dots, \Gamma_m$ be clauses in N such that

$$\Gamma_1, \dots, \Gamma_m, C_1, \dots, C_n \models C.$$

Then the conclusion of the ordered resolution inference is implied by the clauses

$$\Gamma_1, \dots, \Gamma_m, A_1, \dots, A_n, C_1, \dots, C_n,$$

so that the inference is redundant by Proposition 5.6. \square

A resolution inference in which atoms A_1, \dots, A_n are simultaneously resolved can be implemented by a sequence of “partial” inferences, each of which resolves only some of the atoms (with the corresponding side premises).⁵ Partial inferences are reductive and the redundancy of a partial inference therefore implies the redundancy of the original inference. Thus, the above proposition indicates that one may don’t-care non-deterministically choose which of the selected atoms to resolve and also pick the positions in the main premises at which to resolve.

We obtain a corresponding result for self-resolution.

5.11. PROPOSITION. *An ordered self-resolution inference*

$$\frac{\mathcal{D}(A)}{\mathcal{D}(\perp), \mathcal{D}(\top)}$$

is redundant in N , whenever some clause $\Gamma, \mathcal{D}'(\top), \mathcal{D}'(\perp)$ logically follows from clauses in N smaller than \mathcal{D} , where \mathcal{D} can be written as $\Gamma, \mathcal{D}'(A)$ with A occurring in \mathcal{D}' and possibly also in Γ .

By *simple resolution* we mean a variant of general ordered resolution where replacement in the main premise is confined to a single formula, as shown in Figure 8. The refutational completeness of simple resolution follows from the above propositions and Theorem 5.5. Note that the formula F can be chosen don’t-care non-deterministically, as long as it contains at least one occurrence of a resolved atom. Once an inference has been computed for a specific formula F (or otherwise shown to be redundant), then any other inference with a different choice for F is also redundant.

5.4.4. Simplification

An important application of redundancy is in the use of logical equivalences for simplification of clauses. For instance, suppose $N, C' \models C$ and $N, C \models C'$ and $C \succ C'$. Then there is a two-step derivation,

$$N, C \triangleright N, C, C' \triangleright N, C'$$

where the first step is by deduction, as C' is a logical consequence of N, C ; and the second by deletion, as C is rendered redundant by C' . We thus obtain a derived inference rule:

⁵These partial inferences are sound but need not satisfy the restrictions about selection.

Simple ordered resolution with selection

$$\frac{C_1(A_1) \dots C_n(A_n) \quad \mathcal{D}, F(A_1, \dots, A_n)}{C_1(\perp), \dots, C_n(\perp), \mathcal{D}, F(\top, \dots, \top)}$$

where F is a formula such that (i) either A_1, \dots, A_n is selected by S in (\mathcal{D}, F) , or else $S(\mathcal{D}, F)$ is empty, $n = 1$, and A_1 is maximal in (\mathcal{D}, F) , (ii) each atom A_i is maximal in C_i , and (iii) no clause C_i contains a selected atom.

Simple ordered self-resolution

$$\frac{\mathcal{D}, F(A)}{\mathcal{D}, F(\perp), F(\top)}$$

where (i) the atom A is maximal in \mathcal{C} , and (ii) \mathcal{D}, F contains no selected atom.

Figure 8: *Simple ordered resolution SO**Simplification*

$$N, \mathcal{C} \triangleright N, \mathcal{C}'$$

$$\text{if } N, \mathcal{C}' \models \mathcal{C} \text{ and } N, \mathcal{C} \models \mathcal{C}' \text{ and } \mathcal{C} \succ \mathcal{C}'$$

For example,

$$N, (\mathcal{C}, \perp) \triangleright N, \mathcal{C}$$

is a simplification step.

A more interesting case of simplification is the use of object-level equivalences $F \equiv G$ for rewriting. More specifically, we get

$$N, (F \equiv G), \mathcal{C}[F] \triangleright N, (F \equiv G), \mathcal{C}[G] \quad \text{if } \mathcal{C}[F] \succ \mathcal{C}[G]$$

Equivalences occur in many problem domains and very often simplification is the natural way of dealing with them. For example, an equivalence $X \subseteq (Y \cap Z) \equiv [(X \subseteq Y) \wedge (X \subseteq Z)]$ can be used to replace any occurrence of $X \subseteq (Y \cap Z)$ by a conjunction of simpler “subset relations” $X \subseteq Y$ and $X \subseteq Z$. We believe that simplification in this sense may also be the right context for an analysis of the question of “demodulation across argument and literal boundaries,” a research problem posed by Wos [1988].

Meta-level equivalences suitable for simplification can be conveniently described by rewrite systems. For example, by P we denote the set of the following rewrite

rules for elimination of \perp and \top from conjunctions, disjunctions and negations:

$$\begin{array}{ll}
 \alpha \wedge \perp \Rightarrow \perp & \perp \wedge \alpha \Rightarrow \perp \\
 \alpha \wedge \top \Rightarrow \alpha & \top \wedge \alpha \Rightarrow \alpha \\
 \alpha \vee \perp \Rightarrow \alpha & \perp \vee \alpha \Rightarrow \alpha \\
 \alpha \vee \top \Rightarrow \top & \top \vee \alpha \Rightarrow \top \\
 \neg \perp \Rightarrow \top & \neg \top \Rightarrow \perp
 \end{array}$$

If $F \Rightarrow F'$ is a ground instance of a rule in P , then $F \simeq F'$. Furthermore, the rewrite system is contained in any simplification ordering (including lexicographic path orderings). Consequently,

$$N, C \triangleright N, C' \quad \text{if } C \Rightarrow_P^+ C'$$

is a simplification step (for any simplification ordering).

Similar rules for the elimination of \top and \perp can be designed for other connectives, e.g.,

$$\begin{array}{ll}
 \alpha \supset \perp \Rightarrow \neg \alpha & \perp \supset \alpha \Rightarrow \top \\
 \alpha \supset \top \Rightarrow \top & \top \supset \alpha \Rightarrow \alpha \\
 \alpha \equiv \perp \Rightarrow \neg \alpha & \perp \equiv \alpha \Rightarrow \neg \alpha \\
 \alpha \equiv \top \Rightarrow \alpha & \top \equiv \alpha \Rightarrow \alpha
 \end{array}$$

cover implication and equivalence. The simplification rules for eliminating \top and \perp are often directly built into specialized variants of inferences for specific classes (normal forms) of formulas as discussed below.

5.4.5. Normal Forms

Rewrite systems also provide a convenient way of describing various normal forms. Let us briefly discuss negation, conjunctive, and disjunctive normal form. First note that all connectives can be expressed in terms of disjunction, conjunction and negation, as expressed by the rules,

$$\begin{array}{ll}
 \alpha \equiv \beta & \Rightarrow (\alpha \supset \beta) \wedge (\beta \supset \alpha) \\
 \alpha \supset \beta & \Rightarrow \neg \alpha \vee \beta
 \end{array}$$

for the case of implication and equivalence. Termination of these rules can be proved by a lexicographic path ordering in which the symbols to be eliminated (\equiv and \supset in this case) have higher precedence than the other connectives (here \wedge , \vee and \neg).

We may then push negation inside and eliminate double negations:

$$\begin{array}{ll}
 \neg(\alpha \vee \beta) & \Rightarrow \neg \alpha \wedge \neg \beta \\
 \neg(\alpha \wedge \beta) & \Rightarrow \neg \alpha \vee \neg \beta \\
 \neg \neg \alpha & \Rightarrow \alpha
 \end{array}$$

Termination of these rules requires a precedence in which $\neg \succ \vee$ and $\neg \succ \wedge$. The normal forms defined by these rules are also called *negation normal forms*.

From negation normal form we get to *conjunctive normal form* by applying distributivity rules:

$$\begin{aligned}(\alpha \wedge \beta) \vee \gamma &\Rightarrow (\alpha \vee \gamma) \wedge (\beta \vee \gamma) \\ \alpha \vee (\beta \wedge \gamma) &\Rightarrow (\alpha \vee \beta) \wedge (\alpha \vee \gamma)\end{aligned}$$

By C we denote the set consisting of all of the above rules. A lexicographic path ordering, in which $\neg \succ \vee \succ \wedge \succ \top \succ \perp$ and other connectives have higher precedence than \neg , can be used to prove termination of C. We emphasize that we are interested in the existence of normal forms (or “weak normalization,” which is ensured by termination), but not in their uniqueness. Indeed, it is well-known that conjunctive normal forms are not unique. Also, formulas in conjunctive normal form may contain certain “redundancies.” For example,

$$((A \vee A) \wedge (B \vee A)) \wedge ((A \vee \neg B) \wedge (B \vee \neg B))$$

is a conjunctive normal form of $(A \wedge B) \vee (A \wedge \neg B)$, which could be further simplified to

$$(A \wedge (B \vee A)) \wedge (A \vee \neg B).$$

These additional simplifications will be part of the transformation to standard clauses discussed below.

Disjunctive normal forms are obtained by distributing conjunctions over disjunctions:

$$\begin{aligned}(\alpha \vee \beta) \wedge \gamma &\Rightarrow (\alpha \wedge \gamma) \vee (\beta \wedge \gamma) \\ \alpha \wedge (\beta \vee \gamma) &\Rightarrow (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)\end{aligned}$$

To prove termination, we only need to slightly modify the above lexicographic path ordering, so that $\wedge \succ \vee$, instead of $\vee \succ \wedge$.

Inference rules can usually be more efficiently implemented when they need to be applied to clauses in normal form only. Normalization of the clauses is therefore often integrated directly into the computation of inferences, which formally leads to modified inference rules.

A function \mathcal{N} from clauses to finite sets of clauses is called a *normal-form functor* (with respect to \succ) if for each clause C we have that (i) C is equivalent to the conjunction of all clauses in $\mathcal{N}(C)$ and (ii) $C \succeq \mathcal{D}$; and $\mathcal{N}(\mathcal{D}) = \{\mathcal{D}\}$, for each clause \mathcal{D} in $\mathcal{N}(C)$. The clauses \mathcal{D} in $\mathcal{N}(C)$ are also called the *\mathcal{N} -normal forms* of C . If $\mathcal{N}(C) = \{C\}$ the clause C is said to be in *\mathcal{N} -normal form*.

If Γ is an inference system, the *composition* $\mathcal{N} \circ \Gamma$ of Γ with \mathcal{N} is defined to be the set of all inferences

$$\frac{C_1 \dots C_n}{\mathcal{D}}$$

where all premises C_1, \dots, C_n are in \mathcal{N} -normal form, \mathcal{D} is a clause in $\mathcal{N}(C)$, and C is the conclusion of an inference

$$\frac{C_1 \dots C_n}{C}$$

in Γ .

5.12. THEOREM. *Let Γ be an inference system and \mathcal{N} be a normal-form functor with respect to \succ . If Γ has the reduction property with respect to \succ then $\mathcal{N} \circ \Gamma$ has the reduction property with respect to \succ on clauses in \mathcal{N} -normal form.*

PROOF. Suppose Γ has the reduction property with respect to a model functor I and the ordering \succ . Let N be a set of clauses in \mathcal{N} -normal form and \mathcal{C} be a minimal counterexample for I_N in N , which is not a contradiction. Since Γ has the reduction property there is an inference with main premise \mathcal{C} and side premises $\mathcal{C}_1, \dots, \mathcal{C}_n$ in N , the conclusion \mathcal{C}' of which is a smaller counterexample for I_N than \mathcal{C} . Since \mathcal{C}' is equivalent to the conjunction of all the clauses in $\mathcal{N}(\mathcal{C}')$, some clause \mathcal{D}' in $\mathcal{N}(\mathcal{C}')$ must also be a counterexample for I_N . The clause \mathcal{D}' is the conclusion of an inference in $\mathcal{N} \circ \Gamma$ and is a smaller counterexample than \mathcal{C} . \square

This theorem implies that if Γ has the reduction property, then its composition with a normal-form functor \mathcal{N} is refutationally complete for clauses in \mathcal{N} -normal form. Note that it would suffice to require that $\mathcal{C}' \succ \mathcal{D}$, for each clause \mathcal{D} in $\mathcal{N}(\mathcal{C})$, whenever \mathcal{C} is the conclusion of an inference in Γ with main premise \mathcal{C}' in \mathcal{N} -normal form.

5.13. COROLLARY. *If \mathcal{N} is a normal-form functor with respect to \succ and Γ has the reduction property with respect to \succ , then a saturated (with respect to $\mathcal{N} \circ \Gamma$) set of clauses in \mathcal{N} -normal form is either consistent or else contains a contradiction.*

Observe that an inference in Γ from \mathcal{N} -normal forms is redundant if and only if one of the corresponding inferences in $\mathcal{N} \circ \Gamma$ with the same premises is redundant. Moreover a derivation

$$N, \mathcal{C} \triangleright^* N, \mathcal{C}, \mathcal{N}(\mathcal{C}) \triangleright^* N, \mathcal{N}(\mathcal{C})$$

in which one replaces a clause \mathcal{C} by its normal forms, is an instance of simplification.

For example, we may replace a formula by its conjunctive normal form,

$$N, (\mathcal{C}, F) \triangleright N, (\mathcal{C}, F') \quad \text{if } F \Rightarrow_{\mathcal{C}}^! F'$$

and then eliminate conjunctions by a sequence of two deduction steps followed by a deletion,

$$\begin{aligned} & N, (\mathcal{C}, F \wedge G) \\ & \triangleright N, (\mathcal{C}, F \wedge G), (\mathcal{C}, F) \\ & \triangleright N, (\mathcal{C}, F \wedge G), (\mathcal{C}, F), (\mathcal{C}, G) \\ & \triangleright N, (\mathcal{C}, F), (\mathcal{C}, G) \end{aligned}$$

Disjunctions can be eliminated in a similar way,

$$N, (\mathcal{C}, F \vee G) \triangleright N, (\mathcal{C}, F, G),$$

(Binary) ordered resolution with selection

$$\frac{C \vee A \quad D \vee \neg A}{C \vee D}$$

where (i) A is a maximal atom in the side premise and the side premise contains no selected atoms and (ii) the atom A is either selected by S in $D \vee \neg A$, or else $D \vee \neg A$ contains no selected atoms at all and A is maximal and non-positive in D .

Figure 9: *Binary ordered resolution R*

so that any general clause can be reduced to an equivalent finite set of standard clauses. Finally, we may get rid of additional redundancies by applying the simplification rules,

$$\begin{aligned} N, (C, A, \neg A) &\triangleright N, \top \text{ and} \\ N, \top &\triangleright N. \end{aligned}$$

6. Basic Resolution Strategies

Standard resolution can be obtained as a combination of general resolution with a normal-form functor. In this section we outline how several well-known variants of standard resolution can be derived by appropriate settings of the parameters of general resolution. The refutational completeness of these “composite” inference systems follows from Theorem 5.12.

6.1. Binary Resolution

In the theorem proving literature clauses are often represented as sets, rather than multisets, of literals, so that the representation of inferences can be simplified (though there is a trade-off, as the lifting process is more complicated). The transition from multisets to sets can be captured by a simplification rule,

$$N, (C, L, L) \triangleright N, (C, L),$$

which removes duplicate occurrences of literals from a (standard) clause.

Figure 9 depicts a variant of resolution, denoted by R_5^* , that is intended for standard variable-free clauses that represent sets. Self-resolution does not need to be applied to standard clauses, where atoms always have a specific polarity, whether positive or negative; cf. Proposition 5.8.

The following result can be obtained as corollary to Theorems 5.12 and 5.5:

6.1. THEOREM. *Let S be a selection function and \succ be an admissible ordering. If a set of clauses N is saturated up to redundancy under R_S^\succ , then N is unsatisfiable if and only if it contains a contradiction.*

We emphasize that the notion of saturation up to redundancy is flexible enough to cover “mixed” derivations, in which both general and standard resolution inferences appear.

The removal of duplicate literals from a clause can also be expressed as an inference rule:

Positive ordered factoring

$$\frac{C \vee A \vee A}{C \vee A}$$

where A is maximal in C and no atom in C is selected.

Factoring is needed for formulas with variables and therefore its inclusion at the ground level yields a closer correspondence between inferences at the ground and non-ground level. For certain applications a negative version of factoring is needed:

Negative ordered factoring

$$\frac{C \vee \neg A \vee \neg A}{C \vee \neg A}$$

where A is selected or C contains no selected atom and A is maximal in C .

6.2. Hyper-Resolution

Let now S be a selection function which selects one, and only one, negative atom from a clause (if there is a negative occurrence of an atom at all). The only standard clauses from which no atoms can be selected are positive clauses, which contain no negative literals. This kind of selection function results in the following inference rule on standard clauses, which is essentially a special case of general positive ordered resolution, except that the conclusion has been simplified.

Positive ordered resolution

$$\frac{C \vee A \quad D \vee \neg A}{C \vee D}$$

where (i) C is a positive clause with maximal atom A and (ii) the atom A is selected by S in $D \vee \neg A$.

The opposite of this selection function for standard clauses is a “maximal” selection function, where the sequence of all negative atoms in a clause is selected. This leads to the following inference rule:

Ordered resolution with maximal selection

$$\frac{C_1 \vee A_1 \quad \dots \quad C_n \vee A_n \quad D_N \vee D_P}{C_1 \vee \dots \vee C_n \vee D_P}$$

where (i) the clauses C_1, \dots, C_n and D_P are all positive, (ii) no clause C_i contains any of the atoms A_j (not even A_i), (iii) D_N is a negative clause containing all the literals $\neg A_1, \dots, \neg A_n$, and only those literals, and (iv) each atom A_i is maximal in the clause C_i .

Condition (ii) essentially formulates the restriction suggested by Proposition 5.7, though we use the fact that multiple positive occurrences of maximal atoms can be eliminated by ordered positive factoring.

Resolution with maximal selection is closely related to hyper-resolution [Robinson 1965a]. If $C_1 \vee A_1, \dots, C_n \vee A_n$ are positive (standard) clauses, where A_i is maximal in C_i , for all i ; and there exist (standard) clauses D_1, \dots, D_{n+1} , such that D_{i+1} is a resolvent between C_i and D_i on A_i , and D_{n+1} is positive, then

$$\frac{C_1 \vee A_1 \quad \dots \quad C_n \vee A_n \quad D_0}{D_{n+1}}$$

is called a (*positive*) *hyper-resolution* inference. The first n premises are called the *electrons*, the last premise, the *nucleus* of the inference. Since all the premises are standard clauses, the final (positive) clause D_{n+1} (but not the intermediate clauses D_1, \dots, D_n) is independent of the order in which the electrons are listed.

Now consider the premises of an inference by ordered resolution with maximal selection and denote by D_i the simplified version of

$$C_1 \vee \dots \vee C_i \vee D_N[A_1/\top, \dots, A_i/\top] \vee D_P,$$

for $i = 0, 1, \dots, n$. Then D_{i+1} is the conclusion of a (standard) resolution inference on A_i with premises $C_i \vee A_i$ and D_i . In other words, each ordered resolution inference with maximal selection is a hyper-resolution inference. But there are hyper-resolution inferences, such as

$$\frac{A_1 \vee A_2 \quad A_2 \vee A_3 \quad \neg A_1 \vee \neg A_2 \vee A_4}{A_2 \vee A_3 \vee A_4}$$

that are not ordered resolution inferences with maximal selection. (The atom A_2 should not occur in the first premise.) In short, ordered resolution inference with maximal selection is a more restrictive inference system than hyper-resolution.

We should also point out that a resolution inference with maximal selection is redundant if any of the "intermediate resolvents" Γ_i is redundant (cf. Proposition 5.10). This answers an open question in [Wos 1988].

6.3. Boolean Ring-Based Methods

Let AC be the set of (two-way) rewrite rules

$$(\alpha \wedge \beta) \wedge \gamma \Leftrightarrow \alpha \wedge (\beta \wedge \gamma)$$

$$\alpha \wedge \beta \Leftrightarrow \beta \wedge \alpha$$

$$(\alpha \oplus \beta) \oplus \gamma \Leftrightarrow \alpha \oplus (\beta \oplus \gamma)$$

$$\alpha \oplus \beta \Leftrightarrow \beta \oplus \alpha$$

and BR the set of rewrite rules

$$\begin{aligned}
 \neg\alpha &\Rightarrow \alpha \oplus \top \\
 \alpha \vee \beta &\Rightarrow (\alpha \wedge \beta) \oplus (\alpha \oplus \beta) \\
 \alpha \wedge \perp &\Rightarrow \perp \\
 \alpha \wedge \top &\Rightarrow \alpha \\
 \alpha \wedge \alpha &\Rightarrow \alpha \\
 \alpha \oplus \perp &\Rightarrow \alpha \\
 \alpha \oplus \alpha &\Rightarrow \perp \\
 (\alpha \oplus \beta) \wedge \gamma &\Rightarrow (\alpha \wedge \gamma) \oplus (\beta \wedge \gamma)
 \end{aligned}$$

All of these rules describe logical equivalences. Furthermore, the rewrite system BR/AC terminates and the corresponding normal forms, called *BR-normal forms*, are unique up to equivalence under AC [Hsiang 1985]. We denote by $\Phi(F)$ a BR -normal form of F .

Normal forms may also be represented as (possibly empty) *sums*

$$P_1 \oplus P_2 \oplus \cdots \oplus P_n$$

of (pairwise different, possibly empty) *products* of (pairwise different) atoms

$$P_i = A_{i,1} \dots A_{i,n_i}$$

where a product $A_1 A_2 \dots A_n$ represents the formula

$$(A_1 \wedge (A_2 \wedge \cdots (A_{n-1} \wedge A_n) \cdots)),$$

a sum $P_1 \oplus P_2 \oplus \cdots \oplus P_n$ represents the formula

$$(P_1 \oplus (P_2 \oplus \cdots (P_{n-1} \oplus P_n) \cdots)),$$

the empty product denotes \top , and the empty sum denotes \perp . We often need to single out a specific atom and write

$$AP_1 \oplus \dots \oplus AP_m \oplus Q_1 \oplus \dots \oplus Q_n$$

with the understanding that none of the products $P_1, \dots, P_m, Q_1, \dots, Q_n$ contains A . We often use $AP \oplus Q$ as a short form for such a sum of products.

We will now design a refutationally complete calculus for formulas in BR -normal form by composing general ordered resolution with the normal-form functor Φ for generating BR -normal forms. Consider a general ordered resolution inference

$$\frac{AP \oplus Q \quad AP' \oplus Q'}{(\perp \wedge P) \oplus Q, (\top \wedge P') \oplus Q'}$$

Ordered BR-resolution

$$\frac{AP \oplus Q \quad AP' \oplus Q'}{\Phi((Q \oplus \top)(P' \oplus Q') \oplus Q)}$$

where both premises are *BR*-normal forms with maximal atom A .

BR-self-resolution

$$\frac{AP \oplus Q}{\Phi(PQ \oplus P \oplus Q)}$$

where the premise is a *BR*-normal form with maximal atom A .

Figure 10: *Ordered BR Resolution* BR^\succ

in which both premises are sums of products in normal form. The disjunction

$$((\perp \wedge P) \oplus Q) \vee ((\top \wedge P') \oplus Q'),$$

which is logically equivalent to the resolvent, can be simplified to

$$Q \vee (P' \oplus Q').$$

Eliminating the disjunction symbol from the latter formula we get

$$Q(P' \oplus Q') \oplus Q \oplus (P' \oplus Q')$$

which is equivalent to

$$(Q \oplus \top)(P' \oplus Q') \oplus Q.$$

These considerations lead to the inference system BR^\succ displayed in Figure 10. *BR* is monotone with respect to resolution inferences on formulas in *BR*-normal form since the maximal atom A of the two premises does not appear in the conclusion and its *BR*-normal form. (For general clauses, however, we usually have $BR(\mathcal{C}) \succ \mathcal{C}$ if we treat the “,” in \mathcal{C} as disjunction \vee when applying *BR*.) Therefore we may apply Theorem 5.12 and obtain as a consequence of Theorem 5.4:

6.2. THEOREM. *Let N be a set of formulas in *BR*-normal form. If N is saturated with respect to BR and \mathcal{R}^\succ , then N is inconsistent if and only if it contains a contradiction.*

An alternative to ordered *BR*-resolution is a form of critical pair computation for equations between sums of products of atoms. A sum $AP \oplus Q$ with maximal atom A is viewed as an equation $AP \oplus Q \approx \top$ and oriented into a rewrite rule $AP \Rightarrow Q \oplus \top$. Given another rewrite rule $AP' \Rightarrow Q' \oplus \top$ with the same maximal atom, there is a critical pair $P(Q' \oplus \top) \approx P'(Q \oplus \top)$ (between the *AC*-extensions of the rules) that can itself be represented by the polynomial corresponding to $P(Q' \oplus \top) \oplus P'(Q \oplus \top) \oplus \top$. Hence (simple) *BR*-superposition is the following inference:

Simple BR-superposition

$$\frac{AP \oplus Q \quad AP' \oplus Q'}{\Phi(P(Q' \oplus \top) \oplus P'(Q \oplus \top) \oplus \top)}$$

where both premises are *BR*-normal forms with maximal atom A .

This inference rule is sound. In addition, we have

$$A, AP \oplus Q \models P \equiv (Q \oplus \top)$$

and therefore also

$$\begin{aligned} A, AP \oplus Q &\models \\ &P(Q' \oplus \top) \oplus P'(Q \oplus \top) \oplus \top \\ &\equiv (Q \oplus \top)(Q' \oplus \top) \oplus P'(Q \oplus \top) \oplus \top \\ &\equiv (Q \oplus \top)(Q' \oplus \top \oplus P') \oplus \top \\ &\equiv (Q \oplus \top)(P' \oplus Q') \oplus Q. \end{aligned}$$

The calculation shows that if A and $AP \oplus Q$ are true, the equivalence of the conclusions of a *BR*-superposition inference and a *BR*-resolution inference with the same premises follows. For proofs of redundancy of resolution inferences, the resolved atom and the positive premises may be assumed (cf. Proposition 5.6). Hence, the redundancy of a *BR*-superposition inference implies the redundancy of the corresponding ordered *BR*-resolution inference (with the same premises). Denoting by *BRS* the inference system of *BR*-superposition and *BR*-self-resolution, we therefore obtain the following completeness result:

6.3. THEOREM. *Let N be a set of formulas in *BR*-normal form. If N is saturated with respect to *BRS* and $\mathcal{R}^>$, then N is inconsistent if and only if it contains a contradiction.*

Let us next briefly describe how a positive variant of *BR*-resolution can be derived from general positive resolution. First note that a product $A_1 \dots A_n$ is false in the interpretation I_\perp (in which all atoms are false), unless it is the trivial product \top . Consequently, a formula in *BR* normal form is false in I_\perp if, and only if, it does not contain the trivial product \top .

Positive BR-resolution

$$\frac{AP \oplus Q \quad AP' \oplus Q'}{\Phi((Q \oplus \top)(P' \oplus Q') \oplus Q)}$$

where both premises are *BR*-normal forms, the first premise contains no trivial product \top , and A is the maximal atom in the first premise.

On the other hand, a product $A_1 \dots A_n$ is always true in the interpretation I_\top (in which all atoms are true). Thus, a formula in *BR* normal form is false in I_\top if, and only if, it contains an even number of products and is different from \perp . The negative variant of ordered *BR*-resolution is therefore of the following form:

Negative BR-resolution

$$\frac{AP \oplus Q \quad AP' \oplus Q'}{\Phi((Q \oplus \top)(P' \oplus Q') \oplus Q)}$$

where both premises are *BR*-normal forms, the first premise is a sum of an even number of products, and A is the maximal atom in the first premise.

Since positive *BR*-resolution (together with positive *BR*-self-resolution) is simply the composition of general positive resolution $GP^>$ (which is itself an instance of general resolution $O_S^>$ for a specific selection function S) with the normal form functor Φ for *BR*-normal forms, refutational completeness with respect to standard redundancy follows. The same is true, dually, for the negative variant of *BR*-resolution (together with negative *BR*-self-resolution).

It is clear that variants of these inference rules similar to *BR*-superposition are also refutationally complete. Incidentally, the completeness of the latter inference systems was posed as an open problem by Zhang [1994], who introduced negative *BR*-resolution and also mentions its positive dual. In [Zhang 1994] the negative variant of *BR*-resolution is called the “odd strategy,” which may seem odd, given that the inference is characterized by the syntactic restriction that the first premise consist of an *even* number of products. However, Zhang represents the formula $AP \oplus Q$ by an equation $\Phi(AP \oplus Q \oplus \top) \approx \perp$, so that a formula with an even number of products is turned into an equation, the left-hand side of which consists of an odd number of products.

A Boolean ring-based method for first-order theorem proving was first described by Hsiang [1985]. This so-called “N-strategy” is closely related to (standard) negative resolution.⁶ The method applies to equations $A \approx \perp$ where A is a sum of products obtained from the negation of a given *clause*. That is, the initial formulas are assumed to be clauses, the negations of which are translated to sums of products. For instance, the negation of a negative clause $\neg P \vee \neg Q$ is represented by an equation $PQ \approx \perp$ (called an “N-rule”) with a single product of atoms on the left-hand side. The N-strategy is a resolution method with the restriction that one of the premises of each inference be an N-rule. Thus, standard negative resolution is a special case. The N-strategy also allows for simplification by rewriting whereby equations may be transformed so that they no longer represent single standard clauses. However, the method is only complete if rather severe restrictions are imposed on simplification [Zhang 1994]. Thus, the N-strategy in essence more closely resembles standard negative resolution, whereas negative-*BR*-resolution is a true non-clausal method, as shown above.

There are also slightly different approaches that do not derive from non-clausal resolution, but where critical pair computations and other techniques from associative-commutative completion are directly applied to the rewrite system *BR/AC* and polynomial equations; see [Kapur and Narendran 1985] and [Bachmair and Dershowitz 1987] for details.

⁶Various improvements of the original N-strategy have been proposed, e.g., [Müller and Socher-Ambrosius 1988]

7. Refined Techniques for Defining Orderings and Selection Functions

The basic resolution strategies described in the preceding section can be further refined by more elaborate definitions of orderings and selection functions. Key techniques in this regard are renamings of formulas and conservative extensions of a theory. For instance, if a positive literal is renamed to become negative, it can then be selected, so that renaming indirectly allows for a selection of positive literals. A conservative extension of a theory by new symbols, on the other hand, provides more freedom in defining a clause ordering. The semantics of new symbols needs to be specified by suitable formulas. The first step in the theorem proving process then often consists of saturating the collection of all of these “definitions,” whereas later steps correspond to refinements of resolution in the presence of such a pre-saturated subset.

7.1. Renaming and Semantic Resolution

There are several refinements of resolution that are essentially syntactic variants of inference systems described in the previous sections. In particular, this is the case for semantic resolution [Slagle 1967] and hence *set-of-support resolution*. Semantic resolution is defined with respect to a given Herbrand interpretation I :

Semantic resolution

$$\frac{C \vee L \quad D \vee \bar{L}}{C \vee D}$$

where (i) L and \bar{L} are complementary literals, (ii) $C \vee L$ is false in I , and (iii) L is the maximal literal in the first premise.

A well-known refinement of resolution that is covered by semantic resolution relies on a *set of support* [Wos, Robinson and Carson 1965]. Let T (the “theory”) be a satisfiable subset of a set of clauses N . A resolution inference

$$\frac{C \vee L \quad D \vee \bar{L}}{C \vee D}$$

obeys the set-of-support restriction for T if at least one premise is a clause in $N \setminus T$ (the “set of support”).

Let us denote by I_{\perp} the Herbrand interpretation in which all atoms are false, and by I_{\top} the interpretation in which all atoms are true. Semantic resolution with respect to I_{\perp} corresponds to positive resolution;⁷ whereas semantic resolution with respect to I_{\top} has been called *negative resolution*. Positive and negative resolution are dual to each other in that positive resolution is based on the minimal (in a set-theoretic sense) Herbrand interpretation I_{\perp} , whereas negative resolution is based on the maximal Herbrand interpretation I_{\top} . It turns out that semantic resolution

⁷For simplicity, we disregard selection, which has historically not been used in the context of semantics resolution.

with respect to an interpretation I may be seen as a syntactic variant of semantic resolution with respect to any other interpretation I' . We outline how semantic resolution may be mapped to positive resolution via renaming of literals, so that the previous completeness results are applicable.

Let \mathcal{L} be the set of all atoms A_1, A_2, \dots in a given language and \mathcal{L}' be a disjoint set with atoms A'_1, A'_2, \dots . By a *renaming* we mean a mapping ρ from \mathcal{L} to the set of literals over \mathcal{L}' . Most of the renamings ρ we will use satisfy the condition that no two literals $\rho(A_i)$ and $\rho(A_j)$ are complementary. Renamings can be homomorphically extended to clauses and sets of clauses, typically with the modification that a double negation $\neg\neg A$ is replaced by A .

If I' is a Herbrand interpretation over \mathcal{L}' and ρ a renaming from \mathcal{L} to \mathcal{L}' then by I'_ρ we denote the Herbrand interpretation that consists of all atoms A in \mathcal{L} for which $\rho(A)$ is in I' . Therefore, a formula $\rho(F)$ is true in I' if and only if F is true in I'_ρ .

If I is a Herbrand interpretation over \mathcal{L} , the *renaming* ρ_I induced by I is defined as follows: $\rho_I(A_i) = \neg A'_i$, if $A_i \in I$, and $\rho_I(A_i) = A'_i$, if $A_i \notin I$. In other words, ρ_I maps atoms that are false in I to positive literals, and atoms that are true in I to negative literals. For instance, $\rho_I(I_\perp) = \rho_I(\emptyset) = I$. A renamed clause $\rho_I(C)$ is true in I_\perp if and only if C is true in I . Furthermore, if

$$\frac{C \vee L \quad D \vee \bar{L}}{C \vee D}$$

is a semantic resolution inference with respect to I , then

$$\frac{\rho_I(C \vee L) \quad \rho_I(D \vee \bar{L})}{\rho_I(C \vee D)}$$

is a positive resolution inference. In short, semantic resolution with respect to any interpretation I may be viewed as a syntactic variant of positive resolution.

7.1. PROPOSITION. *Let N be a consistent set of standard clauses. Then there exist a renaming ρ and a selection function S such that $\rho(N)$ is saturated up to redundancy by ordered resolution R_S^\triangleright with respect to S and any ordering \succ .*

PROOF. Let I be a model of N and ρ_I be the renaming induced by I . Observe that no clause in $\rho_I(N)$ is positive. Let S be any selection function that chooses at least one atom in each non-positive clause. Then each clause in $\rho_I(N)$ contains a selected atom, which implies that there is no inference by ordered resolution with this selection from $\rho_I(N)$. \square

The significance of this proposition lies in the fact that it asserts the existence of a saturated presentation for any consistent theory, which also implies the refutational completeness of set-of-support resolution.

7.2. Resolution with Free Selection

Selection functions can be used to single out arbitrary negative occurrences of atoms in a clause. The observations in the preceding section show that positive occurrences can be selected indirectly via renaming, though additional clauses defining new symbols need to be introduced. In certain cases, selection can be applied freely to both positive and negative occurrences of atoms.

A *free selection function* is a mapping on clauses that selects exactly one (positive or negative) atom from each (nonempty) clause. By *binary resolution with free selection* we mean a resolution inference in which an atom is resolved only if it is selected in *both* premises. Completeness results for binary resolution with free selection for Horn clauses have been proved by Lynch [1997] and de Nivelle [1996]. We present a simple proof based on a specific kind of renaming.

A *Horn clause* is a standard clause with at most one positive literal. Let N be a set of Horn clause over a language \mathcal{L} and \mathcal{L}' be the set of all expressions $o(A, n)$, where A is an atom in \mathcal{L} , n is a natural number, and o is a new binary symbol. The renaming ρ from \mathcal{L}' to \mathcal{L} is defined by $\rho(o(A, N)) = A$.

7.2. THEOREM. *If a set of Horn clauses N is saturated under binary resolution resolution with free selection, then N is either consistent or else contains the empty clause.*

PROOF. Let N be a set of clauses over \mathcal{L} . We say that a set of clauses N' over \mathcal{L}' encodes N if

- (i) $\rho(C')$ is a clause in N , for each clause C' in N' ,
- (ii) if C is a clause $\neg A_1 \vee \dots \vee \neg A_k$ in N and n_1, \dots, n_k are natural numbers, then $\neg o(A_1, n_1) \vee \dots \vee \neg o(A_k, n_k)$ is a clause in N' , and
- (iii) if C is a clause $\neg A_1 \vee \dots \vee \neg A_k \vee B$ in N and n_1, \dots, n_k are natural numbers, then there is a clause C' in N' of the form $\neg o(A_1, n_1) \vee \dots \vee \neg o(A_k, n_k) \vee o(B, n)$, where $n > \sum_{i=1}^k n_i$.

It can easily be shown that N' is satisfiable whenever N is satisfiable. The converse is also true. For if M is the set of all resolvents from N , then the set $N' \cup K$ encodes $N \cup M$, where K is the set of all resolvents from N' . Therefore, if the empty clause can be derived from N , it can also be derived from N' .

Let now \succ be any admissible ordering on clauses over \mathcal{L}' , such that $o(A, n) \succ o(B, m)$ whenever $n > m$. In addition, if S is a free selection function for N , we define a regular selection function S' for N' , such that S' selects the same negative atom from a clause as S , but selects no atom when S selects a positive atom. There is a correspondence between binary resolution inferences with S from N and ordered resolution inferences with selection function S' from N' , as positive atoms are maximal in the ordering \succ . The refutational completeness of ordered resolution R_S^\exists thus implies the completeness of resolution with free selection, for Horn clauses.

□

A consequence of this theorem is the completeness of SLD-resolution [Kowalski

1974], as used in Prolog, where in each program clause the positive atom (the “head”) is selected and in each negative (or “goal”) clause some negative atom is selected.

Resolution with free selection (together with unrestricted positive and negative factoring) is generally incomplete for non-Horn clauses. For example, consider the inconsistent set of clauses,

$$A \vee \underline{B}, \underline{A} \vee \neg B, \neg \underline{A} \vee B, \neg A \vee \neg \underline{B},$$

where selection is indicated by underlining. By resolution we can derive only tautologies, $B \vee \neg \underline{B}$ and $A \vee \neg \underline{A}$, respectively. Even if these tautologies are not eliminated, with a selection as indicated no new clauses can be derived. But de Nivelle [1996] has shown that free selection for full clauses may cause incompleteness only when every resolution-based refutation requires at least one step of factoring. That is the case in particular for every inconsistent set of binary clauses, as resolution between two binary clauses produces again a binary clause and shorter clauses can only be obtained by factoring.

SL-resolution [Kowalski and Kuehner 1971] is a resolution strategy for full clauses that also employs a rather liberal selection strategy, but we do not know how to justify this strategy within the semantic framework of reducing counterexamples. More specifically, SL-resolution is a refinement of set-of-support resolution where arbitrary, positive or negative atoms may be selected, provided they have been introduced by a theory clause premise of a previous resolution step. It is closely related to model elimination [Loveland 1969] and semantic tableaux. In Sections 8.2 and 8.3 we shall briefly describe how to generalize the linear theorem proving derivations of Section 4.1 to derivation trees. This will allow us to model semantic tableaux and refinements such as model elimination in our framework. We shall in particular see that aspects of selection that cannot be modeled on the semantic level of partial interpretations can often be justified on the level of derivations.

7.3. Conservative Extensions

A straightforward transformation of a formula to clausal form or conjunctive normal form—for instance, via normalization with the rewrite system C—may exponentially increase the size of a formula. Fortunately, there are transformation schemes that preserve the consistency of a formula, but avoid an exponential increase in size. They are based on extending the given language by new predicate symbols and corresponding definitions.

Let N be a set of formulas, F be an expression that occurs as a subformula in N , and L be a literal P or $\neg P$, where the atom P does not appear in any formula in N . We say that a set N' , M is a *regular extension* of N (by L) if (i) N' is obtained from N by replacing one or more occurrences of F by L and (ii) M is logically equivalent to $L \equiv F$. More specifically, we speak of a *positive* (respectively, *negative*) *extension* if only positive (respectively, negative) occurrences of the subformula F are replaced and M is logically equivalent to $L \supset F$ (resp., $F \supset L$). Finally, we say that a set

K is a (*conservative*) *extension* of N if it is obtained from N by a sequence of one or more (regular, positive, and/or negative) extensions.

7.3. PROPOSITION. *If K is an extension of N , then K is consistent if, and only if, N is consistent.*

PROOF. Let I be a model of $N[F]$ and L be a literal P or $\neg P$, where P is not contained in N . If L and F have the same truth value in I , then I is a model of any (regular, positive, or negative) extension of $N[F]$ by L . If L and F have different truth values in I , define I' to be the interpretation $I \cup \{P\}$, if $P \notin I$, and $I \setminus \{P\}$, if $P \in I$. Then L and F have the same truth value in I' (since P does not occur in F), and I' is a model of any (regular, positive, or negative) extension of $N[F]$ by L . This implies that consistency is preserved by any conservative extension.

For the other direction, suppose first that $K = N'$, M is a regular extension of N by L and I is a model of a K . Then L and F have the same truth value in I and, hence, N' and $N[F]$ also have the same truth value in I . Since I is a model of N' , it has to be a model of $N[F]$ as well.

If I is a model of a positive extension N' , M of $N[F]$ by L , then it is a model of N' and of $L \supset F$. Again, if L and F have the same truth value in I , then $N[F]$ is true in I . If L and F have different truth values in I , then L must be false, and F true, in I . Since N' results from replacing positive occurrences of F in N , for any such occurrence $C[F]$ within a clause C of N the clause $C[F/\top]$ is a tautology, which implies that $C[F]$ is true in I . Clauses in N in which F is not replaced also occur in N' and, hence, are true in I by assumption. Thus, I is a model of N .

The case of negative extensions is handled in a similar way. In sum, this implies that if a regular, positive or negative extension of N is consistent, so is N itself. \square

We illustrate the use of the extension principle by showing how a formula can be converted to an equivalent set of standard clauses so that the size increases only by a constant factor, cf., Tseitin [1970].

Let $E_P(L \wedge L')$ be a set of three standard clauses,

$$\neg P \vee L, \neg P \vee L', P \vee \bar{L} \vee \bar{L}',$$

where L and \bar{L} , and also L' and \bar{L}' , are complementary pairs of literals. Similarly, let $E_P(L \vee L')$ be the set

$$\neg P \vee L \vee L', P \vee \bar{L}, P \vee \bar{L}';$$

$E_P(L \supset L')$ the set

$$\neg P \vee \bar{L} \vee L', P \vee L, P \vee \bar{L}';$$

and $E_P(L \equiv L')$ the set

$$\neg P \vee \bar{L} \vee L', \neg P \vee L \vee \bar{L}', P \vee L \vee L', P \vee \bar{L} \vee \bar{L}'.$$

We have the following logical equivalences:

$$E_P(L \wedge L') \simeq P \equiv (L \wedge L')$$

$$E_P(L \vee L') \simeq P \equiv (L \vee L')$$

$$E_P(L \supset L') \simeq P \equiv (L \supset L')$$

$$E_P(L \equiv L') \simeq P \equiv (L \equiv L')$$

Let now N be a finite set of clauses, where we assume for simplicity that all formulas are in negation normal form. (The transformation to negation normal form increases the size of a formula by a constant factor only.) If N is not a set of standard clauses, it must contain a subformula $L \circ L'$, where \circ is one of the connectives \wedge, \vee, \supset or \equiv and L and L' are literals. Then $N[P_{L \circ L'}, E_{P_{L \circ L'}}(L \circ L')]$ is an extension of $N[L \circ L']$, where $P_{L \circ L'}$ is a new predicate symbol. Each extension step eliminates at least one occurrence of a binary connective, so that we eventually end up with a set of standard clauses that is consistent if, and only if, the initial set N is consistent. In the worst case each occurrence of a logical connective in the initial formula has to be replaced by a new atom and at most four additional clauses, each with no more than three literals. Thus, the size of the initial set N may increase only by a constant factor.

Plaisted and Greenbaum [1986] have presented a refinement of this transformation scheme in which the polarities of abbreviated formulas $L \circ L'$ are considered so that for a positive [negative] $L \circ L'$ a positive [negative] extension by $P_{L \circ L'}$ is generated. They also discuss how to automatically extend an ordering to the new predicate symbols in such a way that symbols that represent small formulas are preferred in ordered inferences.

7.4. Lock Resolution

Extension results in interesting variations of resolution, such as lock resolution [Boyer 1971], which can essentially be encoded by positive hyper-resolution.

Lock resolution is applied to standard clauses in which each occurrence of a literal has been assigned a positive integer, called a *lock index*. For example, in the following set N_0 of four clauses,

$${}_8A \quad \vee \quad {}_7B,$$

$${}_6\neg A \quad \vee \quad {}_5\neg B,$$

$${}_4B \quad \vee \quad {}_3\neg A,$$

$${}_2\neg B \quad \vee \quad {}_1A$$

each literal occurrence has been assigned a unique index, but in general different literal occurrences may be assigned the same index. The lock restriction states that only literals with a maximal index must be resolved.⁸ More formally, we have the

⁸We have departed from the original definition in [Boyer 1971], which restricts resolution to minimal literals, so as to avoid confusion with ordered resolution, which resolves maximal literals.

Lock resolution

$$\frac{C \vee_i A \quad \neg_j A \vee D}{C \vee D}$$

where no literal in C has a larger index than i , and no literal in D has a larger index than j .

Figure 11: *Lock resolution*

inference rule given in Figure 11. For example,

$$\frac{{}_8A \vee {}_7B \quad {}_6\neg A \vee {}_5\neg B}{{}_7B \vee {}_5\neg B}$$

is a lock resolution inference, but

$$\frac{{}_8A \vee {}_7B \quad {}_4B \vee {}_3\neg A}{{}_7B \vee {}_4B}$$

is not.

Let $N = C_1, \dots, C_n$ be a finite set of standard clauses (with lock indices). For each clause

$$C_i = {}_{l_{i,1}}A_{i,1} \vee \dots \vee {}_{l_{i,k_i}}A_{i,k_i} \vee {}_{l_{i,k_i+1}}\neg B_{i,k_i+1} \vee \dots \vee {}_{l_{i,k_i+m_i}}\neg B_{i,k_i+m_i}$$

let C'_i be the (renamed) clause

$$C_i = P_{i,1} \vee \dots \vee P_{i,k_i} \vee P_{i,k_i+1} \vee \dots \vee P_{i,k_i+m_i}$$

where the $P_{i,j}$ are new predicate constants not occurring in N , and let M_i be the set of all clauses $\neg P_{i,j} \vee A_{i,j}$, where $1 \leq j \leq k_i$, and $\neg P_{i,k_i+l} \vee \neg B_{i,k_i+l}$, where $1 \leq l \leq m_i$. We also say that $P_{i,j}$ *encodes* the corresponding literal, $A_{i,j}$ or $\neg B_{i,j}$, respectively. We assume that any two of the $P_{i,j}$ are identical if, and only if, the encoded literals are identical and their associated lock indices are the same. Finally, let N' be the (renamed) set C'_1, \dots, C'_n and M be the set M_1, \dots, M_n . The clauses in M are called *definitions*.

Note that a clause $\neg P_{i,j} \vee A_{i,j}$ is logically equivalent to the implication $P_{i,j} \supset A_{i,j}$, and $\neg P_{i,j} \vee \neg B_{i,j}$ is logically equivalent to $P_{i,j} \supset \neg B_{i,j}$. Thus, N', M is an extension of N .

For example, from the matrix N_0 above we get a renamed matrix

$$\begin{array}{lcl} P_{1,1} & \vee & P_{1,2} \\ P_{2,1} & \vee & P_{2,2} \\ P_{3,1} & \vee & P_{3,2} \\ P_{4,1} & \vee & P_{4,2} \end{array}$$

where the predicate constants $P_{i,j}$ are defined by this matrix M :

$$\begin{array}{ll} \neg P_{1,1} \vee A & \neg P_{1,2} \vee B \\ \neg P_{2,1} \vee \neg A & \neg P_{2,2} \vee \neg B \\ \neg P_{3,1} \vee B & \neg P_{3,2} \vee \neg A \\ \neg P_{4,1} \vee \neg B & \neg P_{4,2} \vee A \end{array}$$

Let now \succ be an ordering in which (i) $A_{i,j} \succ P_{i,j}$ and $B_{i,j} \succ P_{i,j}$, for all i, j , and (ii) $P_{i,j} \succ P_{i',j'}$ if, and only if, the lock index $l_{i,j}$ (associated with the literal encoded by $P_{i,j}$) is greater than the lock index $l_{i',j'}$ (associated with the literal encoded by $P_{i',j'}$). We then saturate the set M under ordered resolution. This results, with the given ordering, in the elimination of the “old” atoms $A_{i,j}$, that is, the result is a set M, K , where K consists of all two-element negative clauses $\neg P_{i,j} \vee \neg P_{k,l}$, such that $P_{i,j}$ and $P_{k,l}$ encode complementary literals. The clauses in K are called *connections*.

For the above example we obtain the following connections:

$$\begin{array}{ll} \neg P_{1,1} \vee \neg P_{2,1} & \neg P_{1,2} \vee \neg P_{2,2} \\ \neg P_{1,1} \vee \neg P_{3,2} & \neg P_{1,2} \vee \neg P_{4,1} \\ \neg P_{3,1} \vee \neg P_{4,1} & \neg P_{4,2} \vee \neg P_{2,1} \\ \neg P_{3,1} \vee \neg P_{2,2} & \neg P_{4,2} \vee \neg P_{3,2} \end{array}$$

For instance, the connection $\neg P_{1,1} \vee \neg P_{3,2}$ indicates that in the original matrix N_0 the first literal in the first clause is complementary to the second literal in the third clause.

The set N', M, K is partially saturated in the sense that all inferences with premises from M (i.e., definitions) are redundant in this context. If we use a selection function that selects both literals in a connection, then any possible ordered resolution inference with this selection must be of the form

$$\frac{C \vee P_{i,j} \quad D \vee P_{k,l} \quad \neg P_{i,j} \vee \neg P_{k,l}}{C \vee D}$$

where $P_{i,j}$ is strictly maximal in the first positive premise and $P_{k,l}$ is strictly maximal in the second positive premise. The negative premise is a connection and the conclusion is again a positive clause. In other words, these are positive hyper-resolution inferences with connections as nucleus. The ordering restrictions guarantee that these hyper-resolution inferences encode lock resolution inferences. More precisely, if $C' \vee L$ denotes the clause obtained from $C \vee P_{i,j}$ by replacing each atom by the literal it encodes, and $D' \vee \bar{L}$ is obtained in the same way from $D \vee P_{k,l}$, then

$$\frac{C' \vee L \quad D' \vee \bar{L}}{C' \vee D'}$$

is a lock resolution inference. Conversely, each lock resolution inference is encoded by a positive hyper-resolution inference of the above form. In sum, there is a one-to-one correspondence between hyper-resolution inferences (with two renamed clauses

as negative premises and a connection as positive premise) and lock resolution (on the original clauses).

For example, the hyper-resolution inference

$$\frac{P_{3,1} \vee P_{3,2} \quad P_{4,1} \vee P_{4,2} \quad \neg P_{3,1} \vee \neg P_{4,1}}{P_{3,2} \vee P_{4,2}}$$

encodes the lock resolution inference

$$\frac{{}_4B \vee {}_3\neg A \quad {}_2\neg B \vee {}_1A}{{}_3\neg A \vee {}_1A}$$

We should emphasize that lock resolution, as described above, needs to be combined with factoring (either as inference or as simplification in derivations). The following “syntactic” version of factoring is sufficient:

$$\frac{C, {}_iL, {}_iL}{{}_iL}$$

where L is a literal, and no literal in C has a greater index than i . The correspondence between inferences on original and encoded formulas is not one-to-one in this case. Positive, ordered factoring on encoded formulas correspond to positive or negative lock factoring inferences on the original clauses. The correspondence becomes one-to-one if the same predicate symbol is used to encode different occurrences of an indexed literal.

Boyer [Boyer 1971] describes lock resolution in terms of “semantic factoring,” where a literal ${}_iL$ is deleted from a clause that contains a literal ${}_jL$ with $j > i$, but also mentions the syntactic variant. Semantic factoring is preferable for ground clauses, whereas syntactic factoring is probably a better choice for clauses with variables.

Boyer also mentions that lock resolution is not compatible with tautology deletion. For example, the two lock resolution inferences we have shown at the beginning of this section are the only ones from premises in N_0 . In each case, the conclusion is a tautology. If these inferences were regarded as redundant, then N_0 would be saturated up to redundancy. The calculus would thus be incomplete, as N_0 is inconsistent and contains no contradiction. This observation appears to contradict the fact that our completeness results cover redundancy. But redundancy in our sense has to be applied to encoded clauses and inferences and we can see from the example that the encoding, $P_{3,2} \vee P_{4,2}$, of the tautology ${}_3\neg A \vee {}_1A$ is not itself a tautology and therefore is not redundant.

In some applications of lock resolution as a decision procedure for certain decidable fragments of first-order logic one needs to be able to decrease lock indices don't-care non-deterministically. We will show how to justify such steps within the framework of standard redundancy. A *lock index reduction clause* is a clause of the form $\neg Q \vee P$, where P and Q are “lock symbols” that encode the same original literal L and $P \succ Q$. (We may assume that for each literal L over the given language and lock index i we have a lock symbol Q encoding the indexed literal ${}_iL$, together

with the defining clause for Q .) Lock index reduction clauses encode tautologies. They may be viewed as implications $Q \supset P$, to be applied backwards to decrease lock indices. As the positive literal is maximal, the only possible inferences with a lock reduction clause are of the form

$$\frac{\neg Q \vee P \quad \neg P \vee \neg P'}{\neg Q \vee \neg P'}$$

where a connection in K is the main premise and another connection is the conclusion. (As these partial hyper-inferences are, therefore, redundant we need not consider any "complete" hyper-inferences from $\neg Q \vee \neg P'$ in which one of the side premises is a renamed clause in N' .) It is sound to add these connections initially and no additional resolution inferences result therefrom. The transition

Lock index reduction

$$N, C \vee_i L \triangleright N, C \vee_i L, C \vee_j L \triangleright N, C \vee_j L$$

is admissible in theorem proving derivations based on lock resolution, if $i > j$ and if C contains a literal $_k L'$ such that $k > i$. (The encoding of $C \vee_j L$ is consistency-preserving. Moreover, if the encoding of N contains the lock index reduction clause corresponding to the encodings of the indicated occurrences of $_j L$ and $_i L$, respectively, the encoding of $C \vee_i L$ becomes redundant, provided the lock index reduction clause is be smaller than the encoding of $C \vee_i L$, which is the case if C contains a literal $_k L'$ with $k > i$.)

The condition, that C contain a suitable literal $_k L'$, can be dispensed with when lock index reduction takes place immediately after a clause has been derived in an inference. In the following theorem, the "redundancy" of an inference on original clauses refers to the standard redundancy of the corresponding hyper-resolution inference on the encoded clauses.

7.4. THEOREM. *Let*

$$\frac{C \vee_i A \quad _j \neg A \vee D}{C \vee D}$$

be an inference by lock resolution. The inference is redundant whenever the inference

$$\frac{C \vee_i A \quad _j \neg A \vee D}{C' \vee D'}$$

is redundant, where C' and D' result from C and D , respectively, by decreasing the lock indices of some of the literals.

PROOF. The lock indices in $C \vee D$ are smaller than or equal to the maximum of i and j . Therefore the lock index reduction clauses that are needed to justify the encoding of the implication $(C' \vee D') \supset (C \vee D)$ are smaller than the main premise in the encoding of the inference (which is a connection between $_i A$ and $_j \neg A$). \square

A related result has been obtained by de Nivelle [1996], who also applied it to derive completeness proofs for ordered resolution based on certain so-called “non-liftable” orderings.

In sum, lock resolution is essentially ordered resolution, where initial *occurrences* of atoms (denoted by the labels) are ordered. Accordingly, selection is free in initial clauses. However, since labels are inherited, selection is not free on derived clauses.

7.5. The Inverse Method

The inverse method was proposed by Maslov [1964]. Its basic inference rules are formulated in terms of a given set of (generalized) disjunctions of conjunctions of formulas. In our description of the method we follow Lifschitz [1989], where the method is formulated for disjunctions $G_1 \vee \dots \vee G_k$ of conjunctions $G_i = L_{i,1} \wedge \dots \wedge L_{i,m_i}$ of literals $L_{i,j}$. The disjunctions have been called “super-clauses”, and the conjunctions G_i , “super-literals” in [Lifschitz 1989]. The negation $\neg G_i$ of a super-literal, which is equivalent to a standard clause, is denoted by \overline{G}_i . Given a set of input super-clauses, an *S-clause* is any standard clause that is logically equivalent to a disjunction of negated super-literals \overline{G}_i in the input. For example, given the input

$$\begin{aligned} & (P \wedge \neg Q) \vee (R \wedge T) \\ & \neg P \vee Q \\ & \neg R \end{aligned}$$

the input super-literals are the conjunctions

$$P \wedge \neg Q, R \wedge T, \neg P, Q, \neg R,$$

and their negations are the clauses

$$\neg P \vee Q, \neg R \vee \neg T, P, \neg Q, R.$$

Forming, for instance, the disjunction consisting of the negated first and third super-literals yields the clause

$$\neg P \vee Q \vee P$$

which is a tautology. They represent connections between complementary literals in the input super-clauses.

The inverse method consists of the following two inference rules:

Type A inferences

$$\frac{}{C}$$

where C is any *S-clause* which is a tautology.

Type B inferences

$$\frac{E_1 \vee \overline{G}_1 \quad \dots \quad E_k \vee \overline{G}_k}{E_1 \vee \dots \vee E_k}$$

where $G_1 \vee \dots \vee G_k$ is an input super-clause, and where the premises are S -clauses.

Clearly, the conclusion of a type B inference is again an S -clause. In [Lifschitz 1989] factoring is built into the set notation for clauses.

We will show how to encode this standard version of the inverse method by positive hyper-resolution. The encoding will be similar to the encoding of lock resolution. Let N be a set of formulas F_1, \dots, F_n , where each F_i is a disjunction $G_{i,1} \vee \dots \vee G_{i,m_i}$ of conjunctions $G_{i,j}$ of literals. For each conjunction $G_{i,j}$ we introduce a propositional constant $P_{i,j}$ that does not occur in N , and denote by $M_{i,j}$ a standard clause logically equivalent to $\neg P_{i,j} \supset G_{i,j}$. By M we denote the set of all clauses $M_{1,1}, \dots, M_{n,m_n}$. Let C_i be the clause $\neg P_{i,1} \vee \dots \vee \neg P_{i,m_i}$ and N' be the set of standard clauses C_1, \dots, C_n . Then N', M is an extension of N . (Note that it is sufficient to introduce one constant $P_{i,j}$ for all occurrences of a formula $G_{i,j}$; we need not introduce different constants for different occurrences of $G_{i,j}$ in N .) In the example N' has the clauses

$$\neg P_{1,1} \vee \neg P_{1,2} \quad (1)$$

$$\neg P_{2,1} \vee \neg P_{2,2} \quad (2)$$

$$\neg P_{3,1} \quad (3)$$

with implications

$$\neg P_{1,1} \supset P \wedge \neg Q$$

$$\neg P_{1,2} \supset R \wedge T$$

$$\neg P_{2,1} \supset \neg P$$

$$\neg P_{2,2} \supset Q$$

$$\neg P_{3,1} \supset \neg R$$

and, hence, a set M of clauses,

$$P_{1,1} \vee P$$

$$P_{1,1} \vee \neg Q$$

$$P_{1,2} \vee R$$

$$P_{1,2} \vee T$$

$$P_{2,1} \vee \neg P$$

$$P_{2,2} \vee Q$$

$$P_{3,1} \vee \neg R \quad .$$

As can be observed from the example, in $\neg P_{i,j} \supset G_{i,j}$ (the "definition" of the literal $\neg P_{i,j}$), if $G_{i,j}$ is a conjunction $L_{i,j}^1 \wedge \dots \wedge L_{i,j}^{m_{i,j}}$, then the implication is logically equivalent to the set of binary clauses $P_{i,j} \vee L_{i,j}^1, \dots, P_{i,j} \vee L_{i,j}^{m_{i,j}}$. Let \succ be an admissible ordering in which all new atoms $P_{i,j}$ are smaller than all old atoms occurring in N . If we saturate M under ordered resolution, the result is a set M, K , where K consists of positive clauses of the form $P_{i,j} \vee P_{i',j'}$. The clauses in K encode

the tautologies that can be obtained by "Type A" inferences. In the example, K consists of the clauses

$$P_{1,1} \vee P_{2,1} \quad (4)$$

$$P_{1,1} \vee P_{2,2} \quad (5)$$

$$P_{1,2} \vee P_{3,1} \quad (6).$$

Let us also use a selection function that selects all negative literals in a clause. Then the non-redundant resolution inferences during saturation of N', M, K are positive hyper-resolution inferences of the form

$$\frac{D_1 \vee L_1 \quad \dots \quad D_n \vee L_n \quad \neg L_1 \vee \dots \vee \neg L_n}{D_1 \vee \dots \vee D_n}$$

where the positive premises are positive clauses (initially from K) and the negative premise is from N' . The conclusion is again a positive clause. These inferences correspond to "Type B" inferences (The negative premise $\neg L_1 \vee \dots \vee \neg L_n$ encodes one of the formulas in the original set N .) Conversely, any "Type B" inference can be translated into a positive hyper-resolution inference of this form. In short, we have established a one-to-one correspondence between the (standard version of the) inverse method and positive hyper-resolution. For refutational completeness, ordered factoring for positive clauses has to be added.

In the example one derives a contradiction by the following series of type B inferences:

(7)	$P_{1,2}$	[(6) into (3)]
(8)	$P_{2,1}$	[(4) and (7) into (1)]
(9)	$P_{1,1}$	[(5) and (8) into (2)]
(10)	\perp	[(7) and (9) into (1)]

Maslov's super-clauses represent a particular (non-standard) clausal normal form. Specializing simple ordered resolution SO to super-clauses would result in an inference

$$\frac{\mathcal{C}, A \wedge G \quad \mathcal{D}, \neg A \wedge H}{\mathcal{C}[A/\perp], \mathcal{D}}$$

which is related to (a sequence of two) type B inferences of the inverse method but with slight differences in the way multiple occurrences of the resolved atom A are handled.

7.6. Ordered Theory Resolution

Theory resolution, a concept introduced by Stickel [1985], refers to resolution inferences that have been specially designed for a given consistent set of clauses T , called the *theory*. (Clauses not in T are also called *goal clauses*.) A minimal requirement for a theory resolution calculus is that explicit inferences within the theory be

excluded, so that T needs to be saturated in an appropriate manner. In Section 7.1 we have shown that by renaming one can always obtain a saturated presentation for a consistent set of standard clauses. In fact, set-of-support resolution may be viewed as an instance of theory resolution, though the presentation of T underlying set-of-support resolution is trivially saturated: renaming renders theory clauses non-positive and selection of the non-positive parts therefore permits only inferences in which at least one premise is a goal clause. No non-trivial consequences of T are explicitly represented. More powerful instances of theory resolution can be obtained if the theory T is saturated in a nontrivial way.

Let us first illustrate some of the technical issues. Consider a theory consisting of a single clause specifying that p is a transitive predicate. Note that every (non-ground) binary resolution inference with the transitivity clause $p(x, y) \wedge p(y, z) \supset p(x, z)$ will introduce an extra variable in the resolvent that is not present in the other premise of the inference. Furthermore, unification is no effective filter for inferences as any arbitrary atom $p(s, t)$ can be unified, say, with $p(x, y)$. These issues can be addressed by hyper-inferences in which two literals of the transitivity clause are resolved simultaneously. Then no new variables are introduced and the corresponding unification problem is non-trivial due to the fact that any two literals of the transitivity clause share a common variable. The strategy, admitted by the general theory of ordered resolution, of selecting the two negative literals avoids inferences between theory clauses and generates no extra variables, but is not very useful in practice as it leads to an enumeration of the entire transitive closure of p and, hence, is not goal-oriented enough. A better approach is to select the two literals that contain the maximal of the three (instantiated) *terms* x , y , and z (with respect to a given well-founded ordering on ground terms). The resulting inferences would, on the ground level, eliminate the maximal term from any “reachability” problem involving the transitive predicate p :⁹

Ordered chaining, positive

$$\frac{C \vee p(s, t) \quad D \vee p(t, u) \quad p(s, t) \wedge p(t, u) \supset p(s, u)}{C \vee D \vee p(s, u)}$$

where (i) $t \succ s$, $t \succ u$, (ii) $p(s, t)$ is strictly maximal with respect to C , and (iii) $p(t, u)$ is strictly maximal with respect to D

Ordered chaining, negative (I)

$$\frac{C \vee p(s, t) \quad D \vee \neg p(s, u) \quad p(s, t) \wedge p(t, u) \supset p(s, u)}{C \vee D \vee \neg p(t, u)}$$

where (i) $s \succ t$, $s \succ u$, (ii) $p(s, t)$ is strictly maximal with respect to C , and (iii) $p(s, u)$ is maximal with respect to D .

⁹We present the ground versions of inference systems, but will use clauses with variables in examples.

Ordered chaining, negative (II)

$$\frac{C \vee p(t, u) \quad D \vee \neg p(s, u) \quad p(s, t) \wedge p(t, u) \supset p(s, u)}{C \vee D \vee \neg p(s, t)}$$

where (i) $u \succ t$, $u \succeq s$, (ii) $p(t, u)$ is strictly maximal with respect to C , and (iii) $p(s, u)$ is maximal with respect to D .

This inference system, together with binary ordered resolution and factoring for non-transitivity clauses, turns out to be refutationally complete (for suitable orderings), but involves the selection of non-maximal positive literals in the transitivity clause. The concept of ordered theory resolution which we describe next is centered around more flexible selection strategies for theory clauses. It comes as no surprise that this, again, implicitly involves specific forms of renamings. These inferences will be based on (total) term orderings that have to be extended in an appropriate way to (usually only partial) orderings on atomic formulas.

Let \succ be a *partial*, well-founded ordering on (ground) atoms and T be a set of (ground) clauses. We will consider a calculus in which all maximal atoms in a theory clause from T are resolved simultaneously by hyper-inferences and where inferences between theory clauses are redundant. In the case of transitivity, a suitable ordering can be defined in terms of a well-founded, total ordering on ground terms: $p(s, t) \succ p(s', t')$ if, and only if, either (i) $\max(s, t) \succ \max(s', t')$ (in the term ordering) or (ii) $\max(s, t) = \max(s', t')$ and $s = t$, but $s' \neq t'$. Note that two atoms are uncomparable if they have the same maximal term as one argument and different, non-maximal terms as the other argument.

The description of theory resolution will be facilitated by writing literals as *signed atoms* $+A$ and $-A$, where a *sign* is either “+” or “-”. The signed atom $+A$ denotes the positive literal A , while $-A$ denotes the negative literal $\neg A$. The two signs are considered to be complements of each other. We use the letters σ and τ to denote signs and denote by $\bar{\sigma}$ the complement of σ .

By ordered theory resolution with respect to a theory T , an ordering \succ on atoms, and a total and well-founded extension \succ' of \succ , we mean the following inference rule.

Ordered theory resolution

$$\frac{C_1 \vee \sigma_1 A_1 \quad \dots \quad C_k \vee \sigma_k A_k}{C_1 \vee \dots \vee C_k \vee D}$$

where no premise is in T , but T contains a clause $\bar{\sigma}_1 A_1 \vee \dots \vee \bar{\sigma}_k A_k \vee D$ such that (i) $B \succeq' A_i$ for no atom B in C_i , (ii) the atoms A_i are pairwise incomparable with respect to \succ , and (iii) for each atom B in D there exists an i such that $A_i \succ B$.

In essence, an ordered theory resolution corresponds to a hyper-resolution of goal clauses into a theory clause that resolves all the maximal (with respect to \succ) atoms of the theory clause simultaneously. A (smaller) residuum of non-maximal atoms

from the theory clause forms part of the conclusion of the inference. Only maximal atoms in goal clauses are resolved, where maximality is determined by the admissible extension \succ' of the partial ordering \succ .

The combination of (i) ordered theory resolution and (ii) the restriction of ordered resolution and (positive and negative) ordered factoring to goal clauses, is refutationally complete for presentations that contain sufficiently many logical consequences of the given theory. A key to the completeness proof is to employ a signature extension and renaming so that (i) the maximal atoms in theory clauses become negative literals and, hence, selectable and (ii) goal clauses become positive. This leads to positive hyper-resolution inferences with goal clauses as electrons (and conclusions) and theory clauses as nucleus.

For each propositional symbol A in T we introduce two new symbols, denoted by A_+ and A_- , respectively.¹⁰ If \succ' is any extension of \succ , we order these new atoms as follows: $A_\sigma \succ' B_\tau$ if $A \succ' B$. The intended semantics of the new atoms is captured by *positive and negative connections*, $A_+ \vee A_-$ and $\neg A_+ \vee \neg A_-$, respectively. By K , we denote the set of all connections; by K_- the set of all negative connections; and by K_+ the set of all positive connections.

If C is a clause in T of the form

$$C = \sigma_1 A^1 \vee \dots \vee \sigma_k A^k \vee \tau_1 B^1 \vee \dots \vee \tau_m B^m$$

where the atoms A_i are pairwise incomparable under \succ and each atom B^j is smaller than some atom A^i , then by $\rho(C)$ we denote the renamed clause

$$\neg A_{\sigma_1}^1 \vee \dots \vee \neg A_{\sigma_k}^k \vee B_{\tau_1}^1 \vee \dots \vee B_{\tau_m}^m.$$

For example, if C is the clause $\neg A \vee \neg B \vee C$ with maximal atoms B and C , then $\rho(C) = \neg B_+ \vee \neg C_- \vee A_-$.

Note that a clause C_0 logically follows from clauses C_1, \dots, C_k if, and only if, $\rho(C_0)$ logically follows from $\rho(C_1), \dots, \rho(C_k)$ and K .

we call T a *saturated theory* if for all clauses $C \vee A \vee \dots \vee A$ and $D \vee \neg A \vee \dots \vee \neg A$ in T such that A is maximal (in \succ) with respect to C and D , but does not occur in C or D , the clause $\rho(C \vee D)$ logically follows from $\rho(T) \cup K_- \cup K_+^{\prec A}$, where $K_+^{\prec A}$ is the set of positive connections $B_+ \vee B_-$ for which $A \succ B$. (Note that according to this definition inferences from premises in which a maximal atom occurs both positively and negatively need not be considered.)

At first sight it may seem strange that arbitrary clauses in $\rho(T)$ are admitted in the above definition. But since ρ renders all maximal literals negative, little can be inferred from renamed theory clauses in the absence of positive connections. In particular, it is impossible to infer $\rho(C \vee D)$ directly from the renamed premises $\rho(C \vee A \vee \dots \vee A) = \neg A_- \vee \dots \vee \neg A_- \vee C'$ and $\rho(D \vee \neg A \vee \dots \vee \neg A) = \neg A_+ \vee \dots \vee \neg A_+ \vee D'$ without the connection $A_+ \vee A_-$. A key condition of the above definition is that the positive connections are restricted to atoms smaller than the resolved atom A .

¹⁰The reader should not confuse the meta-level concept of signed atoms σA with the object-level symbols A_+ and A_- . The former simply served as a means to present theory resolution in a concise manner while the latter will be used to rename the original propositions.

If T consists of all ground instances of the transitivity clause for a predicate symbol p (over any given first-order signature) and the given ordering on atoms is based on a term ordering, then T is a saturated theory in this sense. For instance, consider the inference

$$\frac{p(s, t) \wedge p(t, u) \supset p(s, u) \quad p(s, u) \wedge p(u, v) \supset p(s, v)}{p(s, t) \wedge p(t, u) \wedge p(u, v) \supset p(s, v)}$$

where s is a maximal term and $p(s, u)$ a maximal atom in both premises. If s is a strictly maximal term, then renaming the conclusion yields the clause

$$\neg p(s, t)_+ \vee \neg p(s, v)_- \vee p(t, u)_- \vee p(u, v)_-$$

The set $\rho(T)$ contains two other instances of transitivity:

$$\neg p(s, t)_+ \vee \neg p(s, v)_- \vee p(t, v)_-$$

and

$$\rho(p(t, u) \wedge p(u, v) \supset p(t, v)).$$

The maximal term in the latter clause is smaller than s . Therefore the clause contains only atoms smaller than $p(s, u)$ and hence all connections for its atoms are present in $K_- \cup K_+^{\prec p(s, u)}$. Thus $\rho(T) \cup K_- \cup K_+^{\prec p(s, u)}$ entails $\neg p(s, t)_+ \vee \neg p(s, v)_- \vee p(t, u)_- \vee p(u, v)_-$ which was to be shown. Suppose s is not strictly maximal. If $s = t$ or $s = u$, then one of the premises contains its maximal atom both negatively and positively, so that the inference need not be considered. If $s = v$, then we also have $s = u$ (for otherwise $p(s, u)$ would not be maximal) and the previous case applies.

7.5. LEMMA. *Let T be a saturated theory. If I is a model of $\rho(T) \cup K_-$ then there exists an interpretation I' such that $I \subseteq I'$ and I' is a model of $\rho(T) \cup K$.*

PROOF. Let \succ' be any admissible total extension of \succ . We use induction on \succ' to define, for all atoms A in T , interpretations I'_A and E_A as follows.

$$I'_A = I \cup \bigcup_{A \succ B} E_B.$$

If either A_+ or A_- is in I , then E_A is the empty set. Otherwise, E_A is the set $\{A_-\}$, if $I'_A \cup \{A_-\}$ is a model of $\rho(T)$, or else E_A is the set $\{A_+\}$. Finally,

$$I' = I \cup \bigcup_A E_A.$$

By construction I' is a model of K .

Suppose I' is not a model of $\rho(T)$. Let A be a minimal atom such that $I'_A \cup E_A$ is not a model of $\rho(T)$. Then I'_A is a model of $\rho(T) \cup K_- \cup K_+^{\prec A}$ and $E_A = \{A_+\}$. If it is not possible to extend I'_A by either A_+ or A_- , there exists a clause D' of the

form $\neg A_+ \vee \dots \vee \neg A_+ \vee D_1$ in $\rho(T)$ such that D_1 is false in I'_A , and there also exists a clause C' of the form $\neg A_- \vee \dots \vee \neg A_- \vee C_1$ in $\rho(T)$ such that C_1 is false in I'_A . Moreover we may assume that neither C_1 nor D_1 contain $\neg A_+$ or $\neg A_-$. Suppose that $C \vee A$ and $D \vee \neg A$ are the corresponding clauses in T for which $\rho(C \vee A) = C'$ and $\rho(D \vee \neg A) = D'$, respectively, and consider a resolution inference on A from these. As the theory is saturated, $\rho(C \vee D)$ is entailed by $\rho(T) \cup K_- \cup K_+^{\neg A}$. As I'_A is a model for the latter set of clauses, it also satisfies $\rho(C \vee D)$. The two clauses $C_1 \vee D_1$ and $\rho(C \vee D)$ can differ only modulo renaming of literals. More precisely, a literal $+B_\sigma$ occurring in $C_1 \vee D_1$ might occur as $-B_{\bar{\sigma}}$ in $\rho(C \vee D)$. But this is possible only when $A \succ B$. Therefore, $K_- \cup K_+^{\neg A} \models (\rho(C \vee D) \supset (C_1 \vee D_1))$. In conclusion, $C_1 \vee D_1$ must be true in I'_A , which is a contradiction. \square

Let us sketch how the completeness proof for ordered theory resolution can be completed. Consider the ordered resolution calculus $R^{\succ'}$ with a selection function that selects all negative literals in $\rho(T)$ and K_- , and where \succ' is a well-founded extension of \succ that can itself be extended to a total admissible ordering. The given goal clauses are renamed into purely positive clauses, employing the new symbols. Positive connections are not considered for inferences. Then the only non-redundant inferences, apart from factoring are ordered hyper-resolution inferences with goal clauses as electrons and a nucleus that is either a renamed theory clause or one of the connections in K_- . The former inferences correspond to theory resolution, while the latter represent ordered resolution between two goal clauses. If no contradiction can be derived, the set of the renamed goal clauses plus $\rho(T)$ and K_- is satisfiable. If I is a model of this set, it can be extended by the above lemma to a model of K and $\rho(T)$. The renamed goal clauses are positive, and inferences create positive clauses only, so that the extension of the model also satisfies the latter.

A suitable notion of redundancy for theory resolution is represented by standard redundancy for ordered resolution with selection on the renamed goal and theory clauses, together with the connections K_- . In the presence of renaming, certain clauses which denote tautologies before the transformation need not stay redundant, a similar effect to what we have observed in lock resolution.

If T is saturated with respect to a total ordering \succ , there is no difference between an ordered theory resolution inference and an ordered resolution inference in which one premise is a goal clause and the second premise is a theory clause. A partial ordering may produce shorter residuums in theory resolution inferences. On the other hand one would expect that there are more theories that can be effectively saturated under a total atom ordering. Also note that atoms in non-theory predicates do not have to be renamed at all. Their negative occurrences in the original goal clauses can be freely selected.

Technically, the results in this section strictly extend both the results by Baumgartner [1992] and Bronsard and Reddy [1992]. The motivation of the first paper is the specialization of resolution to theories, whereas the second paper considers questions of decidability for saturated theories. We have seen in the example of a transitive relation that saturation with respect to a partial ordering may help to avoid the introduction of new variables in resolution inferences, which in turn is

usually a requirement for decidability. In Bachmair and Ganzinger [1994] we have employed specific methods from term rewriting to obtain a calculus for dealing with transitive relations that is closely related to the chaining calculus that we have presented above as an instance of ordered theory resolution.

Other related approaches of building domain knowledge into resolution include resolution modulo an equational theory E , where syntactic unification is replaced by E -unification [Plotkin 1972], and constraint resolution [Huet 1972, Bürckert 1990]. In these approaches the setup is hierarchic and semantic in that the theory is represented by the set of its models and inconsistency means the failure of being able to extend any of these models to a model of the goal clauses. Bürckert's [1990] constraint resolution can be viewed as form of ordered theory resolution whenever the theory can be represented by a set of first-order clauses.

8. Global Theorem Proving Methods

8.1. The Davis-Putnam Method

A very effective method for testing the satisfiability of a finite set of variable-free standard clauses is the *Davis-Putnam method* [Davis and Putnam 1960]. This method combines simplification (unit reduction, pure literal detection) with a case analysis on atomic formulas. The original method described by Davis and Putnam can be modeled by self-resolution, combined with normalization to conjunctive form and various simplification techniques. But most implementations are based on a modification of the original method [Davis, Logemann and Loveland 1962]. This modified procedure can be formalized in terms of derivation trees, rather than the single "linear" theorem proving derivations we have used so far. The following rule may be viewed as a "tree expansion" rule:

Splitting

$$N \triangleright N, M_1 \quad | \quad \dots \quad | \quad N, M_k \quad (k \geq 1)$$

where N is satisfiable if and only if one of the N, M_i is satisfiable

The application of splitting to a state N creates k new branches as indicated by the states N, M_i . That is, instead of a single derivation sequence we obtain a possibly infinite, but finitely branching tree, each node of which is labeled by a set of clauses.

Each branch in a derivation tree represents a theorem proving derivation, the limit of which needs to be saturated. More formally, if N_0, N_1, N_2, \dots is a branch in the tree, then its limit N_∞ is defined, as before, as the set $\bigcup_i \bigcap_{j \geq i} N_j$. The initial set N_0 , labeling the root of the tree, is unsatisfiable if and only the limit of each branch is unsatisfiable. If the construction of a derivation tree is fair in the sense that the limit of each branch is saturated up to redundancy, then the set $\bigcup_j N_j$ of all formulas along a branch must contain a contradiction, whenever the limit set N_∞ is unsatisfiable. If a branch contains a contradiction, it need not be expanded further, but can be "closed." There are thus two possibilities: either all branches in a derivation tree can be closed, in which case the initial set N_0 is unsatisfiable; or

else some branch can not be closed, in which case a model for N_0 can be defined via the limit N_∞ of the branch. We should point out that the inference system used for deductive inferences along a branch, and the redundancy criterion that controls deletion steps need not be uniform for all branches. For instance, different orderings may be used to restrict resolution steps along different branches. The key requirement is that each branch represent a fair theorem proving derivation (with respect to a refutationally complete inference system and an associated redundancy criterion).

The following rules, which describe the Davis-Putnam method, are based on specific instances of deduction, deletion and splitting. By \bar{L} we denote the complement $\neg L$ of a literal L , with $\neg\neg A$ simplified to A .

Unit reduction

$$N, L, C \vee \bar{L} \triangleright N, L, C$$

Unit subsumption

$$N, L, C \vee L \triangleright N, L$$

Pure literal extension

$$N \triangleright N, L \quad \text{if } L, \text{ but not } \bar{L}, \text{ occurs in } N, \text{ but not as a unit clause}$$

Tautology deletion

$$N, C \triangleright N \quad \text{if } C \text{ is a tautology}$$

Splitting

$$N \triangleright N, A \mid N, \neg A \quad \text{if the atom } A \text{ occurs in } N$$

These rules are applied in a specific order. Splitting, in particular, is applied only if no other rule is applicable. Pure literal extension may trigger subsequent subsumption steps in which all clauses containing the literal L are eliminated.

It is easy to see that the Davis-Putnam method represents a fair strategy for sets of standard ground clauses. If no rule is applicable to a set of ground clauses N , then N either contains the empty clause or else is a set of literals, no two of which are complementary. In the first case, the set N is inconsistent; in the latter case, no resolution and/or factoring inferences are applicable, so that the set is saturated with respect to standard resolution R_S^\succ for any ordering \succ and hence is consistent. The completeness of the method is, therefore, a consequence of Theorem 6.1. The Davis-Putnam method also terminates, provided the initial set of clauses is finite.

The method can be further improved by simplification and deletion steps such as general subsumption or subsumption resolution (see Section 4.3):

Subsumption resolution

$$N, D \vee L, D \vee C \vee \bar{L} \triangleright N, D \vee L, D \vee C$$

Note that splitting on an atom A , followed by unit reduction with A and $\neg A$ on the two respective branches, is an instance of ordered self-resolution:

Splitting as self-resolution

$$\frac{M, F[A]}{M, F[A/\perp], F[A/\top]}$$

One may therefore adopt a different view and represent a derivation tree by a general clause. More specifically, each formula F in a clause M, F is itself in conjunctive normal form and represents a leaf of a partially expanded derivation tree. A self-resolution inference on an atom A in F represents splitting and unit reduction, as mentioned above. The ordering on atoms is defined “on the fly” in that the atom A that is resolved can be declared as the maximal remaining atom in F . The replacement of A by \perp and \top represents unit reduction. If A occurs with only one polarity in F , the conclusion can be simplified to either $M, F[A/\perp]$ or $M, F[A/\top]$.

It does not seem to be possible to model the above methods as a *linear* theorem proving process based on ordered resolution with selection in a natural way. The order in which atoms are resolved does not depend on a single uniform ordering, but is guided by syntactic characteristics of the involved subformulas. Typically, different orderings will be used for different formulas in a clause (which represent different branches in a derivation tree). But linear theorem proving derivations have to adopt a uniform ordering. (The method originally introduced by Davis and Putnam does employ a single ordering, though.)

On the other hand one can show that in a theorem proving derivation one may, without affecting the general results about how to effectively achieve fairness and, hence, the saturation of its limit, always admit *finitely* many “heureka” steps $N \triangleright N'$ in which one replaces the set N by any set N' that preserves consistency or inconsistency. These observations indicate why the Davis-Putnam procedure does not easily extend to the infinite case of clauses with variables.

8.2. Saturated Semantic Tableaux

In the Davis-Putnam method splitting implicitly depends on tautologies of the form $A \vee \neg A$. *Semantic tableau methods* perform case splits directly on given clauses. For standard variable-free clauses we get two kinds of derivation steps:

Splitting on clauses

$$N, (L_1 \vee \dots \vee L_k) \triangleright N, L_1 \quad | \quad N, L_k$$

Ancestor literal complement

$$N, L, \bar{L} \triangleright \perp$$

Splitting on clauses is formally a combination of splitting and subsumption, where the original clause is eliminated via subsumption after the case split. Tableaux are specific, “regular” derivation trees in that no branch contains two nodes to which the same clause split has been applied. The application of the ancestor literal complement rule closes a branch. Formally, it combines a unit resolution step that generates the empty clause, with subsumption steps by which all other clauses are eliminated. If none of the above rules is applicable to a set N , then N consists either

of the empty clause or else of unit clauses no two of which are complementary. Any strategy that applies the two rules exhaustively (and don't-care nondeterministically) is fair, and the limit of each branch in the derivation tree is closed under resolution. Refutational completeness again follows from Theorem 6.1.

The method can be improved by adding ordering constraints, selection functions and simplification, and exploiting the observation that it is sufficient to saturate the limit of each branch with respect to any (refutationally complete) variant of resolution. To that end we may restrict splitting on clauses so that it correspond to ordered resolution with selection.

Splitting on clauses induced by ordered resolution

$$N, C \vee A \vee \dots \vee A, D \vee \neg A \triangleright N, C, \neg A \quad | \quad N, A, D$$

if $C \vee D$ is an ordered resolvent of $C \vee A \vee \dots \vee A$ and $D \vee \neg A$ with respect to a given ordering and selection function.

In this formulation one does not split a clause into its k individual literals, but implicitly uses the tautology $A \vee \neg A$ to do a split into two branches that allows one to eliminate a clause in each branch. This approach is closely related to the Davis-Putnam method. In practice one difference between the two methods has been in the choice of concrete data structures to represent derivation trees. Tableaux are usually represented in a graphical way so that in an expansion

$$N, L_1 \vee \dots \vee L_k \triangleright N, L_1 \quad | \quad | \quad N, L_k$$

the set N that is common to all nodes is not duplicated, but implicitly shared. Even the initial clause set N_0 can be left implicit, so that only the literals L_i on which case splits are performed are stored explicitly. But then properties such as regularity have to be posed as extra constraints.

When one translates the above splitting rule with ordering restrictions and selection into the more common graphical formulations one obtains in particular a justification of the ordering restrictions for tableaux that have been proved complete by Klingenberg and Hähnle [1994]. Again, it is clear that the ordering \triangleright does not need to be uniform for all branches.

In short, the theoretical machinery of this paper suggests a natural generalization of the notion of a closed tableau to a *saturated tableau*, in which all branches are saturated up to redundancy with respect to one of the refutationally complete resolution calculi.

8.3. Model Elimination

Let us next consider sets of standard clauses of the form T, G , where G is a positive clause $A_1 \vee \dots \vee A_k$ and each clause in T contains at least one negative literal. In fact, every inconsistent set of clauses contains such a subset (up to renaming). If N is inconsistent, then there exists an inconsistent finite subset T', G' , where G' is a single clause and T' is consistent. We obtain a set T, G of the above form by

renaming, as suggested by our discussion of semantic and set-of-support resolution in Section 7.1.

The set T is called the *theory* and G is called the *goal clause*. Let S be a selection function that selects exactly one negative literal in each clause in T . *Model elimination* [Loveland 1969] can be described by the following derivation rules:

Splitting on the goal clause

$$T, G \triangleright T, A_1 \mid \dots \mid T, A_k \\ \text{if } k > 1$$

Expansion with a theory clause

$$N, (\neg A \vee L_1 \vee \dots \vee L_n), A \triangleright N, A, L_1 \mid \dots \mid N, A, L_n \\ \text{if } \neg A \text{ is selected in the indicated (theory) clause and } n \geq 1$$

Ancestor literal complement

$$N, L, \bar{L} \triangleright \perp$$

The expansion rule is a special case of the splitting rule on clauses from Section 8.2, where the branch for $\neg A$ has been omitted, as one can immediately derive \perp from A and $\neg A$ and close the branch. The clause that is split is deleted to avoid a repeated case analysis for the same clause. Note that only theory clauses can be split, as all derived clauses consist of one literal only.

The key refinement in model elimination, as compared to the above semantic tableau method, is that the expansion with a theory clause must be triggered by a literal that is complementary to the selected literal in the clause. The literals that may trigger an expansion can all be traced back to the initial goal clause. In a fully expanded model elimination tree each branch is saturated up to redundancy under resolution with selection. Therefore, an initial clause T, G is inconsistent if and only if each branch in the tree ends with \perp .

The above rules have been formulated with respect to an initial clause T, G , which is a renaming of a subset T', G' of a given clause set. In practice one has to compute with the original clauses T', G' , as a suitable renaming, and thus the selection of literals in theory clauses, is usually not known. In fact, even the right choice of the clause G' from $N \setminus T'$ may have to be guessed. In general, all different possible derivations have to be computed in a non-deterministic fashion.

Let us now return to the earlier question of free selection (cf., Section 7.2) and analyze the correspondence between model elimination trees and resolution derivations. In a model elimination tree, each node except the root is labeled either by the constant \perp or else by a set of clauses that is obtained from the parent node by replacing a non-unit clause by a literal, which we call the *main literal* of such a node. Now take any derivation tree in which at least one expansion step was applied and where none of the leaf nodes contains a pair of complementary literals. Let L_1, \dots, L_m be the main literals of those leaves (from left-to-right) which are different from \perp . Then the clause $L_1 \vee \dots \vee L_m$ can be derived from T, G by linear resolution (where at most one premise in each inference is a clause in T) plus (implicit or explicit) factoring. This observation provides an explanation for the linearity constraints in SL-resolution [Kowalski and Kuehner 1971]. In SL-

resolution one may also select an arbitrary literal that was inherited by a resolvent from the theory clause premise of the resolution step. This simply corresponds, in the context of model elimination, to selecting the leaf where a derivation tree is to be expanded next. Obviously, one may don't-care non-deterministically choose any leaf. The ancestor literal complement steps in model elimination either correspond to implicit factoring or to subsumption resolution steps called "ancestor resolution" in [Kowalski and Kuehner 1971]. Tautology elimination and other redundancy elimination techniques described in the latter paper can be easily modeled by standard redundancy.

In conclusion, there is a close correspondence between model elimination trees and certain resolution derivations with simplification. Resolution methods, such as SL-resolution, that exploit this relationship may combine restrictions on the clause level, that can be justified semantically, with restrictions on the level of derivation trees, that can be justified by proof transformations.

9. First-Order Resolution Methods

9.1. *First-Order Sequents*

We have described resolution for (possibly infinite) sets of variable-free formulas. But resolution methods for propositional logic play only a minor role in practice compared to, say, the Davis-Putnam procedure (cf. Section 8.1) or methods based on ordered binary decision diagrams [Bryant 1992]. One of the main applications of resolution and other saturation-based methods is in automated theorem proving for first-order logic. Before resolution can be applied to a first-order formula, quantifiers have to be eliminated and formulas usually have to be converted to clause form. Sophisticated methods of clausal normal form transformation are described in [Baaz, Egly and Leitsch 2001, Nonnengart and Weidenbach 2001] (Chapters 5 and 6 of this Handbook). Inference rules and redundancy criteria need to be lifted to (quantifier-free) clauses with variables.

9.2. *Lifting of Ordering Constraints*

The key parameters in our description of theorem proving methods are orderings, selection functions, renaming strategies, and simplification and deletion techniques. Once these parameters are taken into account, lifting tends to become less straightforward.

In the literature we find two main techniques for lifting ordering constraints. One possibility is to define a "safe" approximation of ground constraints by extending a (total, well-founded) ground ordering to an (partial, well-founded) ordering on non-ground expressions, such that $E \succ E'$ if and only if $E\sigma \succ E'\sigma$, for all ground substitutions σ . If an inference is intended to capture ground instances for which, say, $E\sigma \succ E'\sigma$, one may add a constraint $E' \not\succ E$ at the non-ground level. For

instance (simple, binary) ordered resolution is approximated on the non-ground level by an inference rule (see also Figure 4)

$$\frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma}$$

where σ is a most general unifier of A and B and (i) $A'\sigma \not\leq A\sigma$, for any A' in C , (ii) C contains no selected literal, and (iii) either $\neg B$ is selected or else $B'\sigma \not\leq B\sigma$, for any atom B' in D . It is better to apply the constraints to expressions to which the unifier has been applied as this may render a more precise approximation of the ordering constraints for the relevant ground instances of the inference. The satisfiability of non-ground constraints is undecidable for many orderings, so that one may have to resort to incomplete, but sound, constraint solvers (which represent a weaker approximation of ground constraints). In Section 4.3 we have proved a lifting lemma for binary ordered resolution with selection for this sort of approximation.

9.3. Constrained Formulas

A second, perhaps more adequate method is based on adding constraints to non-ground expressions, at the object level. A *constraint* restricts the set of ground terms that may substituted for variables. Notations such as

$$C \parallel \gamma$$

are used to denote the set of all those ground instances of a non-ground clause C for which the constraint γ is satisfied. For example, standard (binary) ordered resolution (without selection) can be lifted to an inference

$$\frac{C \vee A \parallel \gamma \quad D \vee \neg B \parallel \delta}{C \vee D \parallel \gamma \wedge \delta \wedge (A > C) \wedge (B \geq D) \wedge (A \approx B)}$$

on constrained non-ground clauses. The resolvent is constrained by the maximality conditions $(A > C) \wedge (B \geq D)$ and the equality constraint $A \approx B$, and also inherits the constraints γ and δ from the premises. The notation has been inspired by constraint logic programming and, in the context of automated theorem proving, formalized by Kirchner, Kirchner and Rusinowitch [1990] and others. Completeness proofs for certain saturation-based theorem proving strategies involving constrained clauses were first obtained by Huet [1972], Bürckert [1990] and Nieuwenhuis and Rubio [1992]. In moving the constraints from the meta-level to the object level, and using constraint inheritance, the ordering restrictions of the ground level are represented in a precise way and in principle no information is lost. Constraints are also significant when finite representations of saturated theories are sought, cf. Section 10.2.

These constraints combine logical and meta-logical restrictions in one expression. For example, in the above resolution inference, $A \approx B$ is the logical constraint that

ensures the soundness of the inference, whereas the ordering constraints characterize meta-logical restrictions on inferences. There is some evidence that formalisms that explicitly separate logical from meta-logical constraints might be more appropriate. A meta-logical constraint can be relaxed without affecting the soundness and completeness of the inference system, and certain simplification strategies may actually require such constraint relaxation. Also, solvers for meta-logical constraints need not be complete. Logical constraints, on the other hand, must not be relaxed and the completeness of the theorem proving process requires that their solvability be decidable. Nieuwenhuis and Rubio [2001] (Chapter 7 of this Handbook) present equational theorem proving methods with constrained formulas in detail.

9.4. Resolution Modulo an Equational Theory

Constraints formalisms often provide a suitable framework for building *equational theories* (on ground atoms) into an inference system. One then considers only formulas constructed from the unique canonical representatives of term equivalence classes, a presentation that is preserved under resolution. For ordered inferences, one needs an ordering that is compatible with the equational theory and well-founded on canonical expressions. Refutational completeness requires that solvability of equality constraints be decidable. In cases where sets of unifiers may be very large (e.g., in AC-unification) or even infinite (e.g., in the case of higher-order unification) constraints are indispensable [Huet 1972, Nieuwenhuis and Rubio 1994, Vigneron 1994].

10. Effective Saturation of First-Order Theories

Saturation up to redundancy terminates for many consistent theories, provided powerful enough simplification and redundancy elimination techniques are employed. In this section we briefly describe some of the applications in which finitely saturated theories play a central role. We intend to demonstrate that saturation can be understood as a (partial) compilation process through which, when it terminates, certain objectives with regard to efficiency can be achieved in an automated way.

10.1. Decision Procedures Based on Resolution

The abstract notion of redundancy is general enough to accommodate virtually all the simplification techniques common in theorem proving. With the right setting of the resolution parameters, most of the known decidable fragments of first-order logic can be decided by saturation up to redundancy. The theory of resolution is therefore a powerful tool for decidability proofs for first-order theories and logics that can be semantically embedded in first-order logic. An early example was given by Joyner Jr. [1976], who showed that the monadic class can be decided by ordered resolution with subsumption and condensation as simplification techniques.

Since then, many other decidable classes have been shown decidable by suitably refined calculi of ordered resolution (and paramodulation): the monadic class with equality [Bachmair, Ganzinger and Waldmann 1993], the Ackermann class with equality [Fermüller and Salzer 1993], a subclass of Maslov's class K [Fermüller, Leitsch, Tammet and Zamov 1993] and various logics for knowledge representation [Fermüller et al. 1993, Hustadt 1999]. Hustadt [1999] appears to have been the first to describe a resolution-based decision procedure for conjunctions of formulas in Maslov's class K , the completeness proof of which makes essential use of the methods presented here, in particular of renaming techniques. Schmidt [1997] proposes a general method for obtaining resolution-based decision procedures for many modal logics, and in particular provides a decision procedure for an interesting fragment of first-order logic, called *path logic*. Fermüller et al. [1993] give a comprehensive overview of earlier work on resolution-based decision methods. A recent survey of methods in this area is presented in [Fermüller, Leitsch, Hustadt and Tammet 2001] (Chapter 25 of this Handbook).

10.2. Automated Complexity Analysis

Motivated by work on "local theories" by McAllester [1993], the relation between saturation and decidability issues has been extended to complexity analysis by Basin and Ganzinger [1996], who showed that the complexity of the ordering used for saturation is directly related to the complexity of the entailment problem for a theory.

Let N be a set of standard clauses (with variables). The (*ground*) *entailment problem* for the *theory* N consists in checking whether or not a *query* C , where C is a variable-free standard clause, is logically implied by N . If N is saturated (up to redundancy) under standard ordered resolution without selection (that is, no atom is selected in any clause and, hence, the resolved atom is maximal in both premises), one can derive upper bounds for the complexity of the entailment problem for N . This requires an ordering on atoms in which for each ground atom A there are only finitely many smaller ground atoms. More precisely, suppose that the *complexity* of an ordering can be bounded by functions f, g in that there are at most $O(f(n))$ ground atoms smaller or equal to some ground atom in a given finite set of atoms of size n (where size refers to the number of symbols in an expression), and such that these smaller atoms can be enumerated in time $O(g(n))$.

10.1. THEOREM ([Basin and Ganzinger 1996]). *Suppose \succ is a partial well-founded ordering on ground atoms of complexity f, g and N is a finite set of Horn clauses that is saturated under ordered resolution up to redundancy with respect to each total, well-founded extension of \succ . Then the entailment problem for N is decidable in time $O(f^k + g)$ where k is a constant that depends only on the theory N .*

In particular, the entailment problem is polynomial, if the ordering is of polynomial complexity.

The key technical result is expressed in the following lemma.

10.2. LEMMA. *Let N be saturated up to redundancy with respect to ordered resolution (without selection) based on a total and well-founded ordering \succ . If C is a standard ground clause then C is a logical consequence of N if and only if C is a logical consequence of those ground instances D of N in which for each atom A in D there exists an atom B in C such that $B \succeq A$.*

Let us sketch the basic ideas underlying the lemma. If N is saturated, then inferences in which both premises are clauses in N are redundant. To decide whether some ground query C is entailed, one negates C , adds the resulting unit clauses to N , and saturates the resulting set by ordered resolution. Clauses generated by ordered inferences with one premise in N and one premise from $\neg C$ cannot generate (ground instances of) clauses in which some atom is bigger than each atom in C . Thus, if C is a logical consequence of N , it already follows from a set of ground instances of N in which all atoms are smaller than, or equal to, an atom in C . By applying dynamic programming techniques of bottom-up computation (N was assumed to be a set of Horn clauses), the result follows.

Sometimes a natural presentation of a theory is not saturated with respect to a desired ordering, but can be finitely saturated, see [Basin and Ganzinger 1996] for examples. In such cases saturation can be viewed as an optimizing compiler that adds sufficiently many “useful” consequences to a theory presentation so as to achieve a certain complexity bound for its entailment problem.

11. Concluding Remarks

The presentation of resolution theorem proving in this chapter is based on a general calculus of ordered resolution with selection for general clauses. We have described the more specialized calculi by viewing them as special cases of general resolution. Four concepts—orderings, selection functions, renaming, and redundancy—are essential in this regard. Orderings of clauses are based on well-founded, partial orderings on atoms. Slagle [1967] attributes the original idea of ordering atoms to Reynolds [1965]. Selection functions select atoms that must be true in interpretations in which the clause is false. The earliest resolution strategy that exploits selection appears to be hyper-resolution [Robinson 1965a]. The don’t-care non-deterministic aspects of selection in resolution and the resulting pruning of resolution search spaces were first recognized by Kowalski and Kuehner [1971]. A fundamental theoretical result is the refutational completeness of this family of calculi in the presence of a certain redundancy criterion based on a well-founded ordering on formulas. The proof applies a variant of the model construction technique that was originally introduced in [Bachmair and Ganzinger 1990]. Ideas related to the notion of candidate model can be found in Brand’s proof of completeness of his equality elimination method [1975] and in the work of Zhang [1988]. The definitions presented here are closely related to [Bachmair and Ganzinger 1990], though there the proofs are technically more difficult in that they deal with the equational case and with counterexample reduction and redundancy simultaneously. The paper by

Pais and Peterson [1991] contains a proof of the refutational completeness of superposition based on similar constructions, but without any mention of redundancy. Standard redundancy exploits the fact that only minimal counterexamples to the candidate model have to be reduced. Semantic properties of these partial interpretations can be exploited to further prune the search space [Ganzinger, Meyer and Weidenbach 1997]. Related ideas also play a central role in the global theorem proving method of ordered semantic hyper-linking [Plaisted and Zhu 2000]. Saturation up to redundancy is not only useful as an effective procedure to proving theorems. Refinements of resolution for specific theories can be justified on the (hypothetical) assumption that the theory be presented in a suitable saturated form.

It has often been pointed out that a weakness of resolution is its lack of goal orientation. Simplification and clause elimination based on redundancy helps ameliorate the problem, but one might also consider possible combinations of resolution with such goal-oriented methods as the sequent calculus or semantic tableaux. Avron [1993] provides some discussion of this problem. Semantic tableaux and variants thereof, including the Davis-Putnam method, model elimination and SL-resolution, can be viewed as tree-like theorem proving processes in which the limits of the individual branches are saturated under ordered resolution with selection. This view may serve as a basis for further investigations of the combination problem.

Acknowledgments

We would like to thank the reviewers of this chapter and numerous other colleagues for their helpful comments and suggestions. This research was supported in part by the National Science Foundation under grants CCR-9510072 and CCR-9902031, the German Science Foundation (Deutsche Forschungsgemeinschaft) under grants Ga 261/4-1 and Ga 261/7-1, and the ESPRIT Basic Research Working Group 22457 (Construction of Computational Logics II).

Bibliography

- AVRON A. [1993], 'Gentzen-type systems, resolution and tableaux', *J. Automated Reasoning* 10(2), 265–281.
- BAAZ M., EGLY U. AND LEITSCH A. [2001], Normal form transformations, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 5, pp. 273–333.
- BACHMAIR L. AND DERSHOWITZ N. [1987], Critical pair criteria for rewriting modulo a congruence, in J. Davenport, ed., 'Proc. EUROCAL 87', Vol. 378 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 452–453.
- BACHMAIR L. AND GANZINGER H. [1990], On restrictions of ordered paramodulation with simplification, in M. Stickel, ed., 'Proc. 10th Int. Conf. on Automated Deduction, Kaiserslautern', Vol. 449 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 427–441.
- BACHMAIR L. AND GANZINGER H. [1994], Rewrite techniques for transitive relations, in 'Proc. 9th IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, pp. 384–393.
- BACHMAIR L., GANZINGER H. AND WALDMANN U. [1993], Superposition with simplification as a decision procedure for the monadic class with equality, in G. Gottlob, A. Leitsch and

- D. Mundici, eds, 'Proc. of Third Kurt Gödel Colloquium, KGC'93', Vol. 713 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 83–96. Revised version of Technical Report MPI-I-93-204.
- BASIN D. AND GANZINGER H. [1996], Complexity analysis based on ordered resolution, in 'Proc. 11th IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, pp. 456–465.
- BAUMGARTNER P. [1992], An ordered theory resolution calculus, in A. Voronkov, ed., 'Proceedings of the International Conference on Logic Programming and Automated Reasoning (LPAR'92)', Vol. 624 of *LNAI*, Springer Verlag, St. Petersburg, Russia, pp. 119–130.
- BOYER R. S. [1971], Locking: A restriction of resolution, PhD thesis, University of Texas at Austin, Austin, TX.
- BRAND D. [1975], 'Proving theorems with the modification method', *SIAM J. Comput.* **4**, 412–430.
- BRONSARD F. AND REDDY U. S. [1992], Reduction techniques for first-order reasoning, in M. Rusinowitch and J.-L. Rémy, eds, 'Conditional Term Rewriting Systems, Third International Workshop', LNCS 656, Springer-Verlag, Pont-à-Mousson, France, pp. 242–256.
- BRYANT R. E. [1992], 'Symbolic boolean manipulation with ordered binary-decision diagrams', *ACM Computing Surveys* **24**(3), 293–318.
- BÜRCKERT H.-J. [1990], A resolution principle for clauses with constraints, in M. Stickel, ed., 'Proc. 10th CADE', Vol. 449 of *Lecture Notes in Computer Science*, pp. 178–192.
- DAVIS M., LOGEMANN G. AND LOVELAND D. [1962], 'A machine program for theorem proving', *Communications ACM* **5**(7), 394–397.
- DAVIS M. AND PUTNAM H. [1960], 'A computing procedure for quantification theory', *J. Association for Computing Machinery* **7**, 201–215.
- DE NIVELLE H. [1996], Ordering refinements of resolution, PhD thesis, Technische Universiteit Delft.
- DESHOWITZ N. [1987], 'Termination of rewriting', *J. Symbolic Computation* **3**, 69–116.
- FERMÜLLER C. G. AND SALZER G. [1993], Ordered paramodulation and resolution as decision procedure, in A. Voronkov, ed., 'Proceedings of the 4th International Conference on Logic Programming and Automated Reasoning (LPAR'93)', Vol. 698 of *LNAI*, Springer Verlag, St. Petersburg, Russia, pp. 122–133.
- FERMÜLLER C., LEITSCH A., HUSTADT U. AND TAMMET T. [2001], Resolution decision procedures, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 25, pp. 1791–1849.
- FERMÜLLER C., LEITSCH A., TAMMET T. AND ZAMOV N. [1993], *Resolution Methods for the Decision Problem*, Vol. 679 of *Lecture Notes in Computer Science*, Springer Verlag.
- GALLIER J. H. [1986], *Logic for Computer Science: Foundations of Automatic Theorem Proving*, Harper and Row.
- GANZINGER H., MEYER C. AND WEIDENBACH C. [1997], Soft typing for ordered resolution, in 'Automated Deduction — CADE'14', Vol. 1249 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin, pp. 321–335.
- GILMORE P. C. [1960], 'A proof method for quantification theory', *IBM J. Res. Develop.* **4**, 28–35.
- HSIANG J. [1985], 'Refutational theorem proving using term-rewriting systems', *Artificial Intelligence* **25**, 255–300.
- HUET G. [1972], Constrained Resolution: A Complete Method for Higher Order Logic, PhD thesis, Case Western Reserve University.
- HUSTADT U. [1999], Resolution-Based Decision Procedures for Subclasses of First-Order Logic, PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Germany.
- JOYNER JR. W. [1976], 'Resolution strategies as decision procedures', *J. Association for Computing Machinery* **23**(3), 398–417.

- KAPUR D. AND NARENDRA P. [1985], An equational approach to theorem proving in first-order predicate calculus, in 'Proc. Ninth International Joint Conference on Artificial Intelligence', Los Angeles, CA, pp. 1146–1153.
- KIRCHNER C., KIRCHNER H. AND RUSINOWITCH M. [1990], 'Deduction with symbolic constraints', *Revue Francaise d'Intelligence Artificielle* 4(3), 9–52. Special issue on automatic deduction.
- KLINGENBECK S. AND HÄHNLE R. [1994], Semantic tableaux with ordering restrictions, in A. Bundy, ed., 'Twelfth International Conference on Automated Deduction', LNAI 814, Springer-Verlag, Nancy, France, pp. 708–722.
- KOWALSKI R. A. [1974], Predicate logic as programming language, in 'Proceedings of the IFIP Congress', Amsterdam, Netherlands, pp. 569–4574.
- KOWALSKI R. A. AND KUEHNER D. [1971], 'Linear resolution with selection function', *Artificial Intelligence* 2, 227–260.
- LIFSCHITZ V. [1989], 'What is the inverse method?', *J. Automated Reasoning* 5(1), 1–24.
- LOVELAND D. W. [1969], 'A simplified format for the model-elimination theorem-proving procedure', *J. Association for Computing Machinery* 16, 349–363.
- LYNCH C. [1997], 'Oriented equational logic is complete', *J. Symbolic Computation* 23(1), 23–45.
- MANNA Z. AND WALDINGER R. [1980], 'A deductive approach to program synthesis', *ACM Transactions on Programming Languages and Systems* 2(1).
- MASLOV S. J. [1964], 'An inverse method for establishing deducibility in the classical predicate calculus', *Dokl. Akad. Nauk SSSR* 159, 1420–1424.
- MCALLESTER D. A. [1993], 'Automated recognition of tractability in inference relations', *J. Association for Computing Machinery* 40(2), 284–303.
- MÜLLER J. AND SOCHER-AMBROSIUS R. [1988], Topics in completion theorem proving, Research Report Memo SEKI-SR-88-13, Univ. Kaiserslautern.
- MURRAY N. V. [1982], 'Completely non-clausal theorem proving', *Artificial Intelligence* 18(1), 67–85.
- NIUWENHUIS R. AND RUBIO A. [1992], Theorem proving with ordering constrained clauses, in 'Automated Deduction — CADE'91', Vol. 607 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 477–491.
- NIUWENHUIS R. AND RUBIO A. [1994], AC-superposition with constraints: No AC-unifiers needed, in 'Proc. 12th International Conference on Automated Deduction', Vol. 814 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 545–559.
- NIUWENHUIS R. AND RUBIO A. [2001], Paramodulation-based theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 7, pp. 371–443.
- NONNENGART A. AND WEIDENBACH C. [2001], Computing small clause normal forms, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 6, pp. 335–367.
- PAIS J. AND PETERSON G. [1991], 'Using forcing to prove completeness of resolution and paramodulation', *J. Symbolic Computation* 11, 3–19.
- PLAISTED D. AND GREENBAUM S. [1986], 'A structure-preserving clause form translation', *JSC* 2, 293–304.
- PLAISTED D. AND ZHU Y. [2000], 'Ordered semantic hyper-linking', *J. Automated Reasoning* 25(3), 167–217.
- PLOTKIN G. [1972], 'Building-in equational theories', *Machine Intelligence* 7.
- REYNOLDS J. C. [1965], Unpublished seminar notes. Stanford University.
- ROBINSON J. A. [1965a], 'Automatic deduction with hyper-resolution', *International J. Computer Mathematics* 1, 227–234.
- ROBINSON J. A. [1965b], 'A machine-oriented logic based on the resolution principle', *J. Association for Computing Machinery* 12, 23–41.

- SCHMIDT R. A. [1997], Resolution is a decision procedure for many propositional modal logics, Research Report MPI-I-97-2-002, Max-Planck-Institut für Informatik, Saarbrücken, Germany. The extended abstract version is to appear in M. Kracht, M. de Rijke, H. Wansing and M. Zakharyashev, editors, *Advances in Modal Logic '96*, CSLI Publications 1997.
- SLAGLE J. R. [1967], 'Automatic theorem proving with renamable and semantic resolution', *J. the ACM* **14**(4), 687–697.
- STICKEL M. E. [1985], Automated deduction by theory resolution, in A. Joshi, ed., 'Proceedings of the 9th International Joint Conference on Artificial Intelligence', Morgan Kaufmann, Los Angeles, CA, pp. 1181–1186.
- TSEITIN G. [1970], On the complexity of derivation in propositional calculus, in 'Seminars in Mathematics V.A. Steklov Math. Institute, Leningrad', Vol. 8, Consultants Bureau, New York-London, pp. 115–125. Reprinted in J. Siekmann and G. Wrightson (eds): *Automation of Reasoning*, Vol. 2, 1983, Springer-Verlag, pp. 466–486.
- VIGNERON L. [1994], Associative-commutative deduction with constraints, in 'Proc. 12th International Conference on Automated Deduction', Vol. 814 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 530–544.
- WOS L. [1988], *Automated Reasoning: 33 Basic Research Problems*, Prentice-Hall, Englewood Cliffs, New Jersey.
- WOS L., ROBINSON G. AND CARSON D. [1965], 'Efficiency and completeness of the set of support strategy in theorem proving', *J. Association for Computing Machinery* **12**, 536–541.
- ZHANG H. [1988], Reduction, superposition and induction: Automated reasoning in an equational logic, PhD thesis, Rensselaer Polytechnic Institute, Schenectady, New York.
- ZHANG H. [1994], 'A new method for the boolean ring based theorem proving', *J. Symbolic Computation* **17**(2), 189–212.

Index

Symbols

\triangleright	36
$BR^>$	63
$GP^>$	53
$O_S^>$	33, 41, 49
P	55
R	59
SO	55
RP	42

A

atom	
maximal	27
negative occurrence	28
positive occurrence	28
selected	33, 47

C

candidate model	29, 38
chaining	79
clause	
<i>S</i> -clause	76
admissible ordering	27
dual	23
general	23
Horn	68
maximal atom	27
persistent	36
positive	52
productive	29, 34, 49
reductive	29
redundant	36, 39
standard	23
subsumption	41
variant	42
clause set	
saturated	26
saturated up to redundancy	37
connection	73
conservative extension	70
constraint	90
contradiction	24
counterexample	31, 38

D

Davis-Putnam method	84
---------------------------	----

E

entailment problem	92
expression	

closed	23
first-order	23
ground	23, 24
instance	24
normal form	25

F

factoring	21
negative, ordered	60
positive, ordered	60

I

inference	
main premise	26
redundant	36, 39
rule	25
side premise	26
inference system	
composition with normal-form	
functor	57
consistency-preserving	25
monotone	40
refutational completeness	26
soundness	25
interpretation	
candidate model	29, 34
Herbrand	24
partial	29
inverse method	
type A inferences	76
type B inferences	76

L

lexicographic path ordering	27
literal	
complementary	23
negative	23
positive	23
selected	33
lock index	71

M

model elimination	88
model functor	38

N

Noetherian induction	26
normal form	
BR	62
\mathcal{N}	57

conjunctive	57
disjunctive	57
negation	56
sum of products	62
normal-form functor	57

O

ordering	
admissible	27
extension	26
reduction ordering	26
rewrite ordering	26
simplification ordering	27
subterm property	26
well-founded	26

P

path logic	92
polarity	47
precedence	27

R

reduction property	38
redundancy	
of clauses	39
of inferences	39
redundancy criterion	35
effective	36
standard	39
standard extension	38
trivial	36
refutation	26
renaming	67
induced	67
resolution	
binary	21
binary with factoring	28
<i>Boolean ring</i>	
negative	65
ordered	63
positive	64
self-resolution	63
superposition	64
<i>general</i>	
ordered self-resolution	49
ordered, with selection	49
positive ordered	53
positive self-resolution	53
simple, ordered self-resolution ..	55
simple, ordered with selection ..	55
<i>hyper-resolution</i>	61
<i>lock resolution</i>	72
resolved formula	28
resolvent	28

<i>semantic resolution</i>	66
<i>set-of-support resolution</i>	66
<i>standard</i>	
binary, ordered, with selection ..	59
first-order	41
negative	66
ordered with selection	33
positive, ordered	60
<i>theory resolution</i>	80
with free selection	68
with maximal selection	60
rewrite relation	24
rewrite system	24
modulo	25
terminating	25

S

saturated up to redundancy	37
selection function	33, 47
free	68
semantic tableau	86
saturated	87
simplification	55
backward reduction	42
backward subsumption	42
forward reduction	42
forward subsumption	42
lock index reduction	75
subsumption resolution	42
tautology deletion	42
subsumption	41
proper	42

T

tautology	24
theorem proving derivation	36
fair	37
for extended inference system	38
limit	36
theory	
saturated	81

Tableaux and Related Methods

Reiner Hähnle

SECOND READERS: Uli Furbach and Philippe de Groote.

Contents

1	Introduction	103
2	Preliminaries	104
2.1	Syntax	104
2.2	Semantics	106
3	The Tableau Method	107
3.1	Informal Introduction	107
3.2	Non-clausal Tableaux with Unification	109
3.2.1	Unifying Notation	109
3.2.2	Tableau Rules	110
3.2.3	Tableau Proofs	110
3.2.4	Tableau Semantics and Soundness	112
3.2.5	Universal and Rigid Variables	114
3.2.6	Binary versus n -ary Rules	116
3.3	From Calculus to Proof Procedure	116
3.4	Tableau Completeness	120
3.5	Proof Representation	121
3.5.1	Trees, Matrices, Connections & Matings	121
3.5.2	Pruning Irrelevant Parts of a Proof	124
3.5.3	Simplification	125
4	Clause Tableaux	125
4.1	Normal Form Computation	126
4.2	Clause Tableau Proofs, Soundness, Completeness	126
4.2.1	Clause Tableaux	126
4.2.2	Soundness and Completeness	127
4.3	Connections	129
4.3.1	Connection Tableaux	129
4.3.2	Weak Connections	131
4.4	Regularity	131
4.5	Orderings and Selection Functions	134
4.5.1	Redundancy and Saturation in Tableaux	134
4.5.2	Tableaux with Selection Function	135
4.5.3	Related Calculi	136
4.5.4	First-Order Issues	137

HANDBOOK OF AUTOMATED REASONING

Edited by Alan Robinson and Andrei Voronkov

© 2001 Elsevier Science Publishers B.V. All rights reserved

4.6	Hyper Tableaux	138
4.6.1	Positive and Semantic Hyper Tableaux	140
4.6.2	First-Order Issues	140
4.6.3	Model Generation	143
4.7	A Destructive and Strongly Complete First-Order Calculus	144
4.8	Tableaux with Cuts and Lemmas	146
4.8.1	Tableaux and Sequent Calculi	148
4.8.2	Tableaux with Lemmas	149
4.8.3	Tableaux with Folding Up	150
4.8.4	Tableaux with Factoring	151
4.8.5	Problems of Strengthening Tableaux	151
5	Tableaux as a Framework	152
5.1	Model Elimination	153
5.2	Linear Resolution	155
5.3	Tableaux and (Disjunctive) Logic Programming	155
5.3.1	Near-Horn Prolog, Simplified Problem Reduction Format	159
5.3.2	Restart Model Elimination	161
5.3.3	Restart Tableaux	162
5.4	Davis-Putnam Procedure, KE method and Stålmarck's proof procedure	162
6	Comparing Calculi	164
7	Historical Remarks & Resources	167
	Bibliography	168
	Notation	176
	Index	177

1. Introduction

Reasoning methods based on tableaux and their relatives gained a lot of attention in the past decade after a long period of near stagnation. One reason is that theoretical and implementational progress finally permitted to build tableau-based theorem provers [Moser, Ibens, Letz, Steinbach, Goller, Schumann and Mayr 1997] that can compete [Sutcliffe and Suttner 1999] with state-of-the-art resolution-based systems, at least for logic without equality. Tableau calculi are also well suited to cooperate with [Ahrendt, Beckert, Hähnle, Menzel, Reif, Schellhorn and Schmitt 1998] interactive theorem provers used for software verification, which are usually based on sequent calculi. Another reason is the increased need for deduction in various non-classical logics for which tableau calculi are a particularly good match.

Today, a large number of refinements of tableau-like calculi aimed at efficient automated proof search are available. In fact, there are so many of them that it has become quite difficult for the non-specialist keep track of the main developments. The difficulty of this task is increased by the plethora of names for closely related systems: connection tableaux, connection method, hyper tableaux, matrices, matings, model elimination, model generation, near-Horn logic programming, SL-resolution all are relatives of each other.

In this paper I introduce the main lines of development of tableau-like calculi, as far as they are relevant for automated reasoning, in a uniform framework. At the same time I work out their mutual relationships and I classify the refinements according to various properties.

Most refinements of tableau calculi are defined and implemented only for clause normal form. Accordingly, after a brief treatment of tableaux for full first-order logic in Section 3, the bulk of the material is presented on the clause level (the transformation of arbitrary formulas into clause normal form is discussed in detail in [Baaz et al. 2001, Nonnengart and Weidenbach 2001] (Chapters 5 and 6 of this Handbook)). In Section 4 the main types of refinements of tableau-like calculi are defined and discussed; in Section 5 a number of related calculi are defined relative to the coordinates introduced in the section before. A brief section on comparison and evaluation of calculi follows. The history of tableau-like proof methods is long and vined. Many key ideas were discovered several times independently. I sketch the major developments in the brief historical Section 7.

It was an editorial decision to handle certain topics closely related to tableaux not in the present chapter. Equality reasoning in sequent and tableau calculi is discussed in [Degtyarev and Voronkov 2001a] (Chapter 10 of this Handbook), material on tableaux for non-classical logics in [Baaz, Fermüller and Salzer 2001, Waaler 2001] (Chapters 20 and 22), implementation techniques for (connection) tableau calculi in [Letz and Stenz 2001] (Chapter 28).

2. Preliminaries

Here some basic ingredients of computational logic are collected. This section cannot replace a proper introduction into logic and elementary issues of theorem proving. I recommend Fitting's [1996] book as background.

2.1. Syntax

A *first-order* signature $\Sigma = \langle P_\Sigma, F_\Sigma \rangle$ consists of a non-empty set P_Σ of predicate symbols and a set F_Σ of function symbols. Each symbol in P_Σ, F_Σ has a fixed non-negative arity. In addition, there is an infinite set Var of *variables*.

Given a signature Σ , the sets \mathcal{T}_Σ of *terms* and \mathcal{A}_Σ of *atoms* over Σ are inductively defined by:

1. Variables and 0-ary function symbols from Σ are terms.
2. If t_1, \dots, t_n are terms, f is a n -ary function symbol from Σ , then $f(t_1, \dots, t_n)$ is a term over Σ .
3. If t_1, \dots, t_n are terms, P is a n -ary predicate symbol from Σ , then $P(t_1, \dots, t_n)$ is an atom over Σ .

The *logical operators* are the connectives \vee (disjunction), \wedge (conjunction) and \neg (negation), the quantifier symbols \forall and \exists , and the constant operators *true* and *false*.

Given a signature Σ , the set \mathcal{L}_Σ of *first-order formulas*¹ over Σ is inductively defined by:

1. *true*, *false* and atoms over Σ are formulas.
2. If ϕ is a formula, then $\neg\phi$ is a formula.
3. If ϕ_1, \dots, ϕ_n , $n > 1$, are formulas none of which is a conjunction (resp., disjunction) formula, then $\phi_1 \wedge \dots \wedge \phi_n$ (resp., $\phi_1 \vee \dots \vee \phi_n$) is a conjunction (disjunction) formula.
4. If ϕ is a formula and $x \in \text{Var}$, then $(\forall x)\phi$ and $(\exists x)\phi$ are formulas. ϕ is called the *scope* of the quantifier $(\forall x)$, resp., of $(\exists x)$.

Formulas that are identical up to associativity of \vee and \wedge are identified. Instead of $(\forall x_1) \dots (\forall x_r)\phi$ write $(\forall x_1, \dots, x_r)\phi$. A *literal* is an atom or a negated atom. In the former case, one speaks of a *positive literal*, otherwise of a *negative literal*.

The *size* of a (set of) formula(s) is the number of symbols occurring in it. Let $||\phi||$ stand for the size of a formula or set of formulas.

A formula is in *negation normal form* (NNF) if each occurrence of the negation symbol in it is part of a literal.

A *ground term* (*atom*, *literal*, *formula*) is a term (atom, literal, formula) that contains no variables. The set of ground terms is abbreviated with \mathcal{T}^0 . A *propositional formula* is, by definition, a ground first-order formula, in which no quantifiers or function symbols occur.

¹Implication and equivalence are considered to be defined operators, i.e., $\phi \rightarrow \psi$ is the same as $\neg\phi \vee \psi$, and $\phi \leftrightarrow \psi$ is the same as $(\phi \wedge \psi) \vee (\neg\phi \vee \neg\psi)$.

The *complement* $\bar{\phi}$ of a formula ϕ is defined by: $\bar{\phi} = \psi$ if ϕ is of the form $\neg\psi$, and $\bar{\phi} = \neg\phi$ otherwise.

An occurrence of a variable x in a formula is called *bound* if x occurs in the scope of a quantifier over x , it is called *free* otherwise. A formula without free variable occurrences is a *sentence*.

A *clause* is either the formula *false* or a sentence of the form $(\forall x_1, \dots, x_n)(L_1 \vee \dots \vee L_m)$, $m \geq 1$, where L_i are literals. For sake of readability the quantifier prefix of first-order clauses is usually not written (but assumed to be present). If $m = 1$, we speak of a *unit clause*. The formula *false* is the *empty clause*. Note that clauses are particular formulas. A formula is in *conjunctive normal form (CNF)* if it is of the form $\bigwedge_{i=1}^r C_i$, where C_i are clauses.

A clause with at most one occurrence of a positive literal is a *Horn clause*. A clause with only positive (negative) literals is a *positive (negative) clause*. A non-empty, positive Horn clause is called a *fact* (note that it must contain exactly one literal), a non-empty, negative Horn clause is called a *query*. Non-empty Horn clauses that are neither facts nor queries are called *rule*.

When C, D are ground clauses $C \subseteq D$ means that every literal of C is also a literal of D ; $L \in D$ expresses that the literal L is a literal of clause D . A clause is a *tautology* if it contains literals of the form p and $\neg p$ for some atom p .

A *substitution* is a mapping $\sigma : \text{Var} \rightarrow \mathcal{T}_\Sigma$. It is extended to terms and (sets of) formulas as follows:

1. $\sigma(c) = c$
2. $\sigma(\text{true}) = \text{true}$, $\sigma(\text{false}) = \text{false}$
3. $\sigma(s(t_1, \dots, t_n)) = s(\sigma(t_1), \dots, \sigma(t_n))$ for $s \in F_\Sigma \cup P_\Sigma$, arity of s is n
4. $\sigma(\phi_1 \bullet \dots \bullet \phi_n) = \sigma(\phi_1) \bullet \dots \bullet \sigma(\phi_n)$ for $\bullet \in \{\wedge, \vee\}$
5. $\sigma(\neg\phi) = \neg\sigma(\phi)$
6. $\sigma((Qx)\phi) = (Qy)\sigma'(\phi)$ for $Q \in \{\forall, \exists\}$, where $\sigma' = \sigma \setminus \{x \mapsto t \mid t \in \mathcal{T}\} \cup \{x \mapsto y\}$ and y is a variable not occurring in $(Qx)\phi$ such that $\sigma(y) = y$
7. $\sigma(\{\phi_1, \dots, \phi_n\}) = \{\sigma(\phi_1), \dots, \sigma(\phi_n)\}$

If $\sigma(x) = x$ for all but finitely many $x \in \text{Var}$ we denote σ by $\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, where $\{x_1, \dots, x_n\}$ are exactly the variables with $\sigma(x_i) = t_i \neq x_i$, and $\sigma(x) = x$ for all other variables. Application of substitutions is usually written postfix, composition of substitutions $\sigma \circ \rho$ is denoted by $\rho\sigma$ (note that $\phi(\sigma \circ \rho) = (\phi\rho)\sigma = \phi\rho\sigma$). When for the substitution σ all $\sigma(x)$ for $x \in V \subseteq \text{Var}$ are ground terms one has a *grounding substitution* for the variables V . A *renaming* for a set of variables $\{x_1, \dots, x_n\}$ is a substitution $\nu = \{x_1 \mapsto y_1, \dots, x_n \mapsto y_n\}$ such that the y_i are new and different variables in the context, where ν appears. An *idempotent substitution* is a substitution, for which $\sigma \circ \sigma = \sigma$.

Let $|S|$ denote the cardinality of a set S . If T is a non-empty set of terms and $|T\sigma| = 1$, then σ is a *unifier* of T . It is a *most general unifier (MGU)* if for all unifiers ρ of T there is a substitution θ such that $\rho = \theta \circ \sigma$. Unifiability of a finite set of terms can be decided in linear time. A unifiable set of terms has always an idempotent MGU. See [Baader and Snyder 2001] (Chapter 8 in this Handbook) for details.

An *instance* or, more precisely, a σ -instance of a clause $C = (\forall x_1, \dots, x_n)(L_1 \vee \dots \vee L_m)$ is a formula $(L_1 \vee \dots \vee L_m)\sigma$, where σ is any substitution. One has a *new instance* of C , if σ is a renaming for $\{x_1, \dots, x_n\}$. When σ is a grounding substitution for $\{x_1, \dots, x_n\}$ one has a *ground instance* of C .

The *subformulas* of a formula ϕ are recursively defined as follows:

1. Every formula is a subformula of itself
2. If $\phi = \phi_1 \bullet \dots \bullet \phi_n$, then any formula of the form $\phi_{i_1} \bullet \dots \bullet \phi_{i_r}$ is a subformula of ϕ , where $\{i_1, \dots, i_r\} \subseteq \{1, \dots, n\}$ and $\bullet \in \{\wedge, \vee\}$
3. If $\phi = \neg\psi$, then ψ is a subformula of ϕ
4. If $\phi = (Qx)\psi$, then ψ is a subformula of ϕ , where $Q \in \{\forall, \exists\}$

If ψ is a subformula of ϕ and $\phi \neq \psi$ then ψ is a *proper subformula* of ϕ . If ψ is a proper subformula of ϕ such that there is no proper subformula ρ of ϕ with ϕ being a proper subformula of ρ , then ψ is an *immediate subformula* of ϕ .

An occurrence of a subformula ρ in $\phi \in \mathcal{L}_\Sigma$ is

1. *positive* if $\phi = \rho$,
2. *negative* (positive) if ϕ is of the form $\neg\psi$ and the occurrence of ρ is positive (negative) in ψ ,
3. positive (negative) if ψ is an immediate subformula of ϕ , but $\phi \neq \neg\psi$, and the occurrence of ρ is positive (negative) in ψ .

2.2. Semantics

Given a first-order signature Σ , a *first-order structure* $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ consists of a non-empty set \mathbf{D} called *domain* and an *interpretation* \mathbf{I} that assigns to each n -ary function symbol $f \in F_\Sigma$ a mapping $\mathbf{I}(f) : \mathbf{D}^n \rightarrow \mathbf{D}$ and to each n -ary predicate symbol $P \in P_\Sigma$ a relation $\mathbf{I}(P) \subseteq 2^{\mathbf{D}^n}$.

A *variable assignment* for a first-order structure \mathbf{M} is a mapping $\mu : \text{Var} \rightarrow \mathbf{D}$. The *d-variant* of μ at x is

$$\mu_x^d(y) = \begin{cases} d & \text{if } x = y \\ \mu(y) & \text{otherwise} \end{cases}$$

For a first-order structure \mathbf{M} over signature Σ with variable assignment μ we define $t^{\mathbf{M}, \mu}$ for all $t \in \mathcal{T}_\Sigma$ inductively:

$$\begin{aligned} x^{\mathbf{M}, \mu} &= \mu(x) \text{ for } x \in \text{Var} \\ f(t_1, \dots, t_n)^{\mathbf{M}, \mu} &= \mathbf{I}(f)(t_1^{\mathbf{M}, \mu}, \dots, t_n^{\mathbf{M}, \mu}) \text{ for } f(t_1, \dots, t_n) \in \mathcal{T}_\Sigma \end{aligned}$$

Truth of formulas $\phi \in \mathcal{L}_\Sigma$ in \mathbf{M} under μ , written $(\mathbf{M}, \mu) \models \phi$, is defined as follows:

$$\begin{aligned} (\mathbf{M}, \mu) &\models \text{true} && \text{for all } \mathbf{M} \text{ and } \mu \\ (\mathbf{M}, \mu) &\models \text{false} && \text{for no } \mathbf{M} \text{ and } \mu \\ (\mathbf{M}, \mu) &\models P(t_1, \dots, t_n) && \text{iff } (t_1^{\mathbf{M}, \mu}, \dots, t_n^{\mathbf{M}, \mu}) \in \mathbf{I}(P) \text{ for } P(t_1, \dots, t_n) \in \mathcal{A}_\Sigma \\ (\mathbf{M}, \mu) &\models \neg\phi && \text{iff not } (\mathbf{M}, \mu) \models \phi \\ (\mathbf{M}, \mu) &\models \phi_1 \wedge \dots \wedge \phi_n && \text{iff } (\mathbf{M}, \mu) \models \phi_i \text{ for all } i \in \{1, \dots, n\} \\ (\mathbf{M}, \mu) &\models \phi_1 \vee \dots \vee \phi_n && \text{iff } (\mathbf{M}, \mu) \models \phi_i \text{ for at least one } i \in \{1, \dots, n\} \end{aligned}$$

$$\begin{array}{ll}
(\mathbf{M}, \mu) \models (\forall x)\phi & \text{iff } (\mathbf{M}, \mu_x^d) \models \phi \text{ for all } d \in \mathbf{D} \\
(\mathbf{M}, \mu) \models (\exists x)\phi & \text{iff } (\mathbf{M}, \mu_x^d) \models \phi \text{ for at least one } d \in \mathbf{D}
\end{array}$$

A formula ϕ is *satisfiable* in \mathbf{M} , if there exists a μ such that $(\mathbf{M}, \mu) \models \phi$. A set of formulas is satisfiable, if each of its members is satisfiable simultaneously under the same variable assignment.

A first-order structure \mathbf{M} over signature Σ is a *model* of a set of formulas $\Psi \subseteq \mathcal{L}_\Sigma$, denoted $\mathbf{M} \models \Psi$, if $(\mathbf{M}, \mu) \models \phi$ for all $\phi \in \Psi$ and variable assignments μ . In the light of this definition, CNF formulas are identified with finite sets of clauses. A sentence ϕ is a *logical consequence* of a set of sentences Ψ , denoted $\Psi \models \phi$ if each model of Ψ is also a model of ϕ . ϕ is *valid*, written $\models \phi$, when each structure \mathbf{M} is a model of ϕ .

A first-order structure $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ over signature Σ is a *term domain structure* if $\mathbf{D} = \mathcal{T}_\Sigma^0$.

2.1. PROPOSITION. *For all sentences $\phi \in \mathcal{L}_\Sigma$ and sets of \mathcal{L}_Σ -sentences Ψ : $\Psi \models_\Sigma \phi$ iff $\Psi \cup \{\neg\phi\}$ is unsatisfiable.*

For skolemization we do not use symbols from F_Σ but from a special infinite set F_{sko} of *Skolem function symbols* that is disjoint from F_Σ ; the extended signature $\langle P_\Sigma, F_\Sigma \cup F_{sko} \rangle$ is denoted with Σ^* .

Here is a variant of the Löwenheim-Skolem theorem that will be needed:

2.2. THEOREM. *If a sentence $\phi \in \mathcal{L}_\Sigma$ is satisfiable, then it has a Σ^* -term model.*

In the following, assume F_Σ contains at least one constant, then $\mathcal{T}_\Sigma^0 \neq \emptyset$. A term domain structure $\langle \mathcal{T}_\Sigma^0, \mathbf{I} \rangle$, where, in addition, $\mathbf{I}(f)(t_1, \dots, t_n) = f(t_1, \dots, t_n)$ for all $f \in F_\Sigma$, is called *Herbrand structure*.

2.3. THEOREM (Herbrand's Theorem). *Assume that ϕ is a sentence of the form $(\forall x_1, \dots, x_r)\psi$, where ψ is quantifier-free; ϕ is unsatisfiable iff there is an $m \geq 1$ and grounding substitutions θ_i for $\{x_1, \dots, x_r\}$ such that $\bigwedge_{i=1}^m (\psi\theta_i)$ is unsatisfiable. The minimal number m , for which this is possible is the Herbrand complexity of ϕ .*

In particular, let S be a finite set of clauses. Then S is unsatisfiable iff there is a finite unsatisfiable set \hat{S} of ground instances of S .

The result is due to Herbrand [1930b]; a proof is, for example, in [Smullyan 1995].

3. The Tableau Method

3.1. Informal Introduction

It is common to view the tableau method as a proof by contradiction and case distinction (this view was already stressed by pioneers Beth [1955] and Hintikka [1955]). More precisely, it allows one to systematically generate subcases until elementary contradictions are reached. Let us go through a small example:

Assume we want to prove the following simple theorem from elementary set theory: for arbitrary sets P, Q, R ,

$$\text{if } \left\{ \begin{array}{ll} (1) & P \neq \emptyset \\ (2) & P \subseteq Q \\ (3) & Q \subseteq R \end{array} \right\} \text{ then } P \cap R \neq \emptyset.$$

The proof is by contradiction: assume $P \cap R = \emptyset$. From (1) we know that there is an element $c \in P$. Now apply (2) to c : if $c \in P$, then $c \in Q$. But we know already that $c \in P$, hence, $c \in Q$. Note that (2) can be seen as an implicit case distinction: either $c \notin P$ or $c \in Q$, where the first case immediately contradicts (1). In the same way, deduce from $c \in Q$ with (3) that $c \in R$. At this point, apply the assumption to c : either $c \notin P$ or $c \notin R$. Both cases yield a contradiction immediately.

The proof is easier to follow, if displayed tree-like as in Figure 1. Observe that case distinctions can be generated schematically depending on the form of their premise. At several points, premises had to be suitably instantiated.

Premises (2), (3), and the assumption are universally quantified, for instance, the assumption says that *for all elements x , x cannot be both an element of P and of R* . In automated theorem proving finding instances is done by *unification*—one tries to find a substitution that produces a contradiction in the current branch of the proof (in the example this is $\{x \mapsto c\}$). In general one needs, of course, to apply a premise more than once during a proof. The number of applications, closely related to Herbrand complexity in Theorem 2.3, cannot be computed in advance (otherwise, first-order logic would be decidable). One of the problems that tableau methods must solve is to systematically enumerate “enough” (this is made precise later) instances of universally quantified formulas.

From premise (1) one obtains existentially quantified expressions saying that P must contain at least one element. In automated theorem proving such witness elements are produced by skolemization: the existentially quantified variable is replaced by a “new” term that has not yet an interpretation (again, this is made precise later).

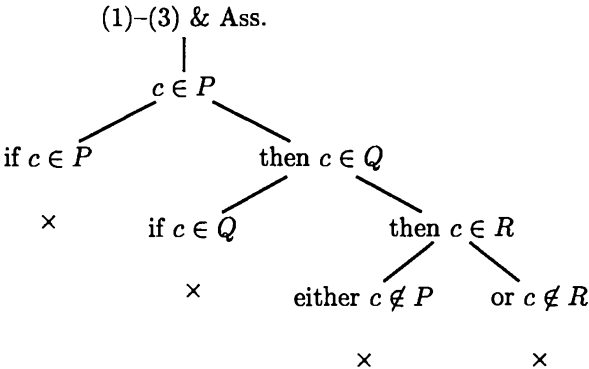


Figure 1: Structure of an informal proof by contradiction and case distinction.

α	$\alpha_1, \dots, \alpha_n$	β	β_1, \dots, β_n
$\phi_1 \wedge \dots \wedge \phi_n$	ϕ_1, \dots, ϕ_n	$\phi_1 \vee \dots \vee \phi_n$	ϕ_1, \dots, ϕ_n
$\neg(\phi_1 \vee \dots \vee \phi_n)$	$\neg\phi_1, \dots, \neg\phi_n$	$\neg(\phi_1 \wedge \dots \wedge \phi_n)$	$\neg\phi_1, \dots, \neg\phi_n$
$\neg\neg\phi$	ϕ		
$\neg\text{false}$	true		
$\neg\text{true}$	false		

γ	γ_1	δ	δ_1
$(\forall x)(\phi(x))$	$\phi(x)$	$\neg(\forall x)(\phi(x))$	$\neg\phi(x)$
$\neg(\exists x)(\phi(x))$	$\neg\phi(x)$	$(\exists x)(\phi(x))$	$\phi(x)$

Table 1: Correspondence between formulas and their types.

3.2. Non-clausal Tableaux with Unification

3.2.1. Unifying Notation

The first step in formalizing the considerations in the previous section is to supply formal rules that tell in which way a formula is analyzed according to its leading connective. Smullyan [1963] and Lis [1960] independently observed² that some work can be saved if non-literal formulas are grouped into types which are treated identically: α for formulas of conjunctive type, β for formulas of disjunctive type, γ for quantified formulas of universal, and δ for quantified formulas of existential type. Correspondence between formulas and their types is summarized in Table 1. By convention, doubly negated formulas and negated logical constants are treated as type α -formulas (with $n = 1$).

The letters α , β , γ , and δ are used to denote formulas of (and only of) the appropriate type. In the case of γ - and δ -formulas the variable x bound by the (top-most) quantifier is made explicit by writing $\gamma(x)$ and $\gamma_1(x)$ (resp., $\delta(x)$ and $\delta_1(x)$); accordingly $\gamma_1(t)$ denotes the result of replacing all occurrences of x in γ_1 by t . Without loss of generality assume that no variable of t occurs in the scope of a quantifier in γ_1 or δ_1 . If necessary, this can be achieved by renaming the bound variables in γ_1 and δ_1 . Associativity of \wedge and \vee justifies conjunctive and disjunctive formulas with an indefinite number of arguments.

Some authors [Lis 1960, Smullyan 1995] prefer to work with *signed formulas*. These are expressions of the form $T\phi$, $F\phi$, where ϕ is a formula. Signed formula tableaux relate more directly to sequent calculi, because T -signed formulas play the rôle of formulas standing left of a sequent arrow while F -signed formulas are on the right (see also Section 4.8.1 below). In classical logic theorem proving there is no particular gain from signs, but in non-classical logics their use is indispensable

²See Section 7; see [Fitting 1999] for a full historical account.

[Beckert and Goré 1997, Hähnle 1999].

3.2.2. Tableau Rules

With the help of unifying notation decomposition rules for arbitrary formulas can be given in a concise way.

In Table 2 expansion rule schemata for the various formula types are given. Premises and conclusions are separated by a horizontal bar, while vertical bars in the conclusion denote different *extensions*. The formulas in an extension are implicitly conjunctively connected, and different extensions are implicitly disjunctively connected. We use n -ary α - and β -rules; for example, when the β -rule is applied to a formula $\psi = \phi_1 \vee \dots \vee \phi_n$, then ψ is broken up into n subformulas (instead of splitting it into two formulas $\phi_1 \vee \dots \vee \phi_r$ and $\phi_{r+1} \vee \dots \vee \phi_n$, $0 < r < n$).

Type γ formulas are simply stripped from their quantifier while the quantified variable is renamed into a variable not occurring elsewhere. Instantiation of free variables is delayed.

The δ -rule is the most technical rule. Its purpose is to replace an existential quantifier with a witness element or Skolem term. It incorporates two important optimizations with respect to the more straightforward rule of [Fitting 1990]: the first is that the choice of the witness element merely depends on the free variables in δ , not on all free variables on the current branch; in addition, the leading function symbol f of the *Skolem term* may be the same for δ -formulas which are identical up to variable renaming, formally:

3.1. DEFINITION. Given a signature $\Sigma = \langle P_\Sigma, F_\Sigma \rangle$, the function *sko* assigns to each $\delta \in \mathcal{L}_\Sigma$ a symbol $sko_\delta \in F_{sko}$ such that (a) $sko_\delta > f$ for all $f \in F_{sko}$ occurring in δ , where $>$ is an arbitrary but fixed ordering on F_{sko} , and (b) for all $\delta, \delta' \in \mathcal{L}_\Sigma$ the symbols sko_δ and $sko'_{\delta'}$ are identical if and only if δ and δ' are identical up to variable renaming (including renaming of the bound variables).

The purpose of condition (a) in the above definition of *sko* is to avoid cycles like: sko_δ occurs in δ' and $sko'_{\delta'}$ occurs in δ .

Both improvements of the δ -rule together have the consequence that its conclusion can be computed locally to the formula δ —no “global” information is required.

Skolemization rules for normal form computation are due to Andrews [1971] and Bibel [1982c]; specific tableau rules seem to appear first in [Brown 1978], they gained wide popularity through [Fitting 1990]. Our δ -rule is from [Beckert, Hähnle and Schmitt 1993], further improvements are possible [Baaz and Fermüller 1995, Giese and Ahrendt 1998, Cantone and Nicolosi Asmundo 1998].

3.2.3. Tableau Proofs

As was hinted at already, tableau proofs are trees whose nodes are formulas that are (sub)goals in the proof and the tree structure gives the logical dependence between them. Assume we want to prove that a set of sentences Φ logically implies a sentence ψ . By Proposition 2.1 this amounts to checking that the set of sentences $\Phi \cup \{\neg\psi\}$ is unsatisfiable.

α	β	$\gamma(x)$	$\delta(x)$
α_1	$\beta_1 \mid \cdots \mid \beta_n$	$\gamma_1(y)$	$\delta_1(sko_\delta(x_1, \dots, x_n))$
\vdots		$y \in \text{Var}$ is new	x_1, \dots, x_n are the
α_n		to the tableau.	free variables in δ .

Table 2: Rule schemata for tableaux with unification.

3.2. DEFINITION. Let Σ be a first-order signature. A *tableau* (over Σ) is a finitely branching tree whose nodes are formulas from \mathcal{L}_Σ . A *branch* in a tableau T is a maximal path in T .³ A *tableau calculus* is a set R of rules each having a set Φ of sentences from \mathcal{L}_Σ and, optionally, a tableau for Φ as premises, and another tableau for Φ as conclusion. For each concrete Φ , the transitive closure of these rules defines a set of *tableaux constructed with R for Φ* .

Our main example of a tableau calculus in the present section are *tableaux with unification*:

3.3. DEFINITION. Given a set Φ of sentences from \mathcal{L}_Σ , a *tableau with unification for Φ* is defined as a tableau constructed with the following rules:

1. The tree consisting of a single node *true* is a tableau for Φ (initialization rule).
2. Let T be a tableau for Φ , B a branch of T , and ψ a formula in $B \cup \Phi$. Consider an arbitrary instance of a tableau rule schema in Table 2 with premise ψ and n extensions. Obtain the tree T' by extending B with n new linear subtrees whose nodes are the formulas in the extensions of the rule instance. Then T' is a tableau for Φ (expansion rule).
3. Let T be a tableau for Φ , B a branch of T , and ψ and ψ' literals in $B \cup \Phi$. If ψ and $\overline{\psi'}$ are unifiable with MGU σ , and T' is constructed by applying σ to all formulas in T (i.e., $T' = T\sigma$), then T' is a tableau for Φ (closure rule).

The last item in this definition incorporates two conditions: first, MGUs are used instead of arbitrary substitutions; second, ψ and ψ' are literals, not arbitrary formulas. The former is crucial, because there are only finitely many MGUs (up to renaming of variables) of formulas and their complements in a finite tableau. If Φ is finite this implies that there are systematic procedures for enumerating the (finite) tableaux for Φ .

Branches in a tableau correspond to different subcases in a proof. Formulas occurring in tableaux with unification may contain free variables, hence, Definition 3.3(3) is required to produce an explicit contradiction in a subcase/branch. A tableau proof is finished when this has been achieved for all branches, formally:

³When no confusion can arise, branches are frequently identified with the set of their nodes (formulas).

3.4. DEFINITION. In a tableau T for a set Φ of sentences a branch B is *closed* iff $B \cup \Phi$ contains a pair $\phi, \neg\phi \in \mathcal{L}_{\Sigma^*}$ of complementary formulas, or *false*; otherwise, it is *open*. A tableau is closed if *all* its branches are closed.

A *tableau proof* for (the unsatisfiability of) a set $\Phi \subset \mathcal{L}_{\Sigma}$ of sentences is a closed tableau T for Φ .

3.5. REMARK. In the previous definition, not only formulas from B , but also from Φ are permitted to participate in branch closure. As a consequence, for example, any tableau for $\Phi = \{\text{false}\}$ is closed. Some authors prefer to define branch closure with respect to only B . In this case an additional tableau construction rule that fetches formulas from Φ and places them on some branch is required. Our version was chosen, because it allows a more uniform presentation of various calculi.

3.6. EXAMPLE. We are now in a position to formalize the introductory example from Section 3.1. Sets P, Q, R are represented by unary predicates P, Q, R : their characteristic functions. Then, over the signature $\Sigma = \langle \{P, Q, R\}, \{\} \rangle$, the claim holds if and only if the set Φ consisting of the following \mathcal{L}_{Σ} -sentences is unsatisfiable:

- (1) $(\exists x)P(x)$
- (2) $(\forall x)(\neg P(x) \vee Q(x))$
- (3) $(\forall x)(\neg Q(x) \vee R(x))$
- (4) $(\forall x)(\neg P(x) \vee \neg R(x))$

Figure 2 shows a tableau T with unification for Φ . The nodes of the tableau are numbered starting from 5 (the numbers 1–4 refer to the formulas in Φ); an expression $[i; j]$ is in front of the i -th node N_i , where j signifies that N_i stems from an expansion rule applied to N_j (respectively, to formula (j) in Φ).

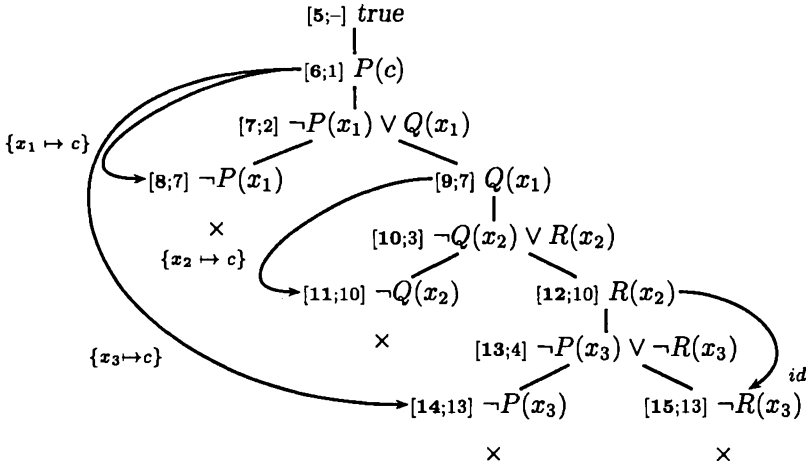
All branches of T can be closed; a closure is indicated by an arc between its complementary literals, labeled with the required MGU. Observe that MGUs are applied to *all* nodes in the tree. For example, the MGU of nodes 12 and 15 is the identity, because x_3 is instantiated during unification of nodes 6 and 14. Convince yourself that the tableau is well-defined.

In the following we say just ‘tableau’ instead of ‘tableau with unification’, if it is clear from the context that the latter is meant.

Occasionally, we speak of the size of a tableau. Formally, the *size* of a tableau is the sum of the sizes of the formulas occurring in it.

3.2.4. Tableau Semantics and Soundness

Since our goal is to use tableaux as a framework for formal proofs, we require to extend semantics from formulas to tableaux. Our guideline here is to ensure that there exists a closed tableau for Φ iff Φ is unsatisfiable. We fixed already that a tableau represents the disjunction of its branches which in turn are considered as conjunctions of their labels. By a standard argument then, the equivalence above is

Figure 2: Tableau proof for Φ from Example 3.6.

reduced to the question whether the tableau construction rules leave tableau satisfiability unaltered. This is a routine matter for all but the δ -rule, which requires some care. Recall that two optimizations were incorporated into this rule (Section 3.2.2): (i) the variables of the Skolem term are restricted to the free variables of δ , (ii) the leading function symbol sko_δ of the Skolem term is not unique in a tableau proof. Tableau semantics must be carefully chosen to reflect these restrictions. To meet (i) it suffices to treat free variables in a tableau essentially as if they were universally quantified.

3.7. DEFINITION. A tableau T for $\Phi \in \mathcal{L}_\Sigma$ is *satisfiable* if there is a structure \mathbf{M} of Φ such that for every variable assignment μ there is a branch B of T with $(\mathbf{M}, \mu) \models B$. In that case we say that \mathbf{M} is a model of T , denoted by $\mathbf{M} \models T$.

For (ii) it is important that Skolem function symbols are interpreted in the “right” way. The most elegant way to achieve this, is to define formula semantics with respect to only such interpretations—let us call them *canonical interpretations*. Of course, one needs to show then that each satisfiable formula can be satisfied by a canonical interpretation.

3.8. DEFINITION. A term domain structure $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ is *canonical* iff for all variable assignments μ and all $\delta(x) \in \mathcal{L}_{\Sigma^*}$: if $(\mathbf{M}, \mu) \models \delta(x)$ then $(\mathbf{M}, \mu) \models \delta_1(sko_\delta(x_1, \dots, x_n))$, where x_1, \dots, x_n are the free variables in δ .

3.9. LEMMA ([Beckert and Hähnle 1998]). *Given a signature Σ , if the set $\Phi \in \mathcal{L}_\Sigma$ of sentences is satisfiable, then there is a canonical structure \mathbf{M}^* over Σ^* such that $\mathbf{M}^* \models \Phi$.*

3.10. COROLLARY. Let M^* be a canonical structure over Σ^* , μ a variable assignment, and $\phi \in \mathcal{L}_{\Sigma^*}$; and let ϕ' be constructed from ϕ by (a) replacing a positive occurrence of some $\delta(x)$ in ϕ by $\delta_1(\text{sko}_{\delta(x)}(x_1, \dots, x_n))$, or by (b) replacing a negative occurrence of $\delta(x)$ in ϕ by $\delta_1(\text{sko}_{\delta(x)}(x_1, \dots, x_n))$, where x_1, \dots, x_n are the free variables in $\delta(x)$. Then $(M^*, \mu) \models \phi$ implies $(M^*, \mu) \models \phi'$.

3.11. LEMMA. Any tableau T with unification constructed for a satisfiable set of \mathcal{L}_{Σ} -sentences is satisfiable.

PROOF. By definition of tableaux with unification, there is a sequence T_1, \dots, T_m ($m > 0$), where $T = T_m$ and T_1 is the initial tableau whose single node is true, and where T_{i+1} is constructed from T_i by applying a single tableau expansion or closure rule. By Lemma 3.9, the input set is satisfied by a canonical structure M^* over Σ^* . By induction on m one proves that M^* satisfies all of T_1, \dots, T_m and hence T . The induction step is easy (see [Fitting 1996] for details) and all cases, but the δ -rule case are straightforward. The latter, however, holds by the corollary. \square

Now assume we have a closed tableau T for a set of \mathcal{L}_{Σ} -sentences Φ . Obviously, no structure and variable assignment can satisfy a closed branch, so T is unsatisfiable. By the preceding lemma, Φ is unsatisfiable as well. This proves:

3.12. THEOREM (Soundness). If there is a tableau proof for a set $\Phi \subset \mathcal{L}_{\Sigma}$ of sentences, then Φ is unsatisfiable.

Completeness is stated and proven in Section 3.4.

3.2.5. Universal and Rigid Variables

In general, different instances of the variables in the scope of a universal quantifier are needed in order to close a branch (or a subtableau). In tableaux with unification the mechanism to achieve this is to apply the γ -rule multiply to generate formula instances with different free variables. It is crucial to note that free variables in tableaux are *not* implicitly universally quantified locally to the branch on which they occur⁴, but are *rigid*: any substitution σ with $\sigma(x) \neq x$ must be applied to *all* occurrences of x in a tableau. Figure 3 shows an unsound tableau proof for the invalid formula $\psi = (\forall x)(P(x) \vee Q(x)) \rightarrow ((\forall x)P(x) \vee (\forall x)Q(x))$ that would be possible if free variables were not handled rigidly.

In some cases, though, it is sound to treat free variables as if they were quantified universally. For example, if we have a tableau for $\Phi = \{\neg P(c) \vee \neg P(d), (\forall x)P(x)\}$ that consists of two branches, one containing $P(x_1)$ and $\neg P(c)$, and the other containing $P(x_1)$ and $\neg P(d)$. This tableau cannot be closed immediately as no single substitution closes both branches. To find a proof, the γ -rule has to be applied again to create another new instance of $(\forall x)P(x)$. In this example, $(\forall x)P(x)$ is a logical consequence of Φ and the formulas already on the tableau (in a sense made

⁴In contrast to this, resolvent clauses in a resolution calculus, for example, are universally quantified.

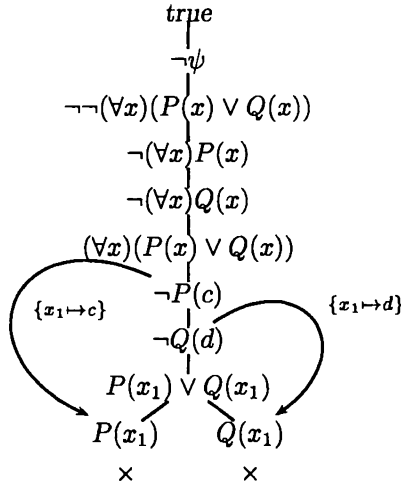


Figure 3: Unsound tableau proof due to non-rigid treatment of free variables.

precise in Definition 3.13), hence, $(\forall x)\phi(x)$ can be added to *each* branch. In this situation, substitutions with differing values for x can be used without destroying soundness of the calculus. The tableau for Φ then would close earlier. Recognizing such situations and exploiting them allows using more general closing substitutions, yields shorter tableau proofs, and in many cases reduces the search space.

3.13. DEFINITION. Suppose ϕ is a formula on a branch B of a tableau T for $\Phi \in \mathcal{L}_\Sigma$. Let T' result from adding $(\forall x)\phi$ to B for some $x \in \text{Var}$. Formula ϕ is called *universal* on B with respect to x if every model of T is also a model of T' .⁵ Denote with $\text{UVar}(\phi, B)$ the variables with respect to which ϕ is universal on B .

Instead of designing a closure rule that takes universal variables into account (Definition 3.3(3)), we generalize the concept of a unifier:

3.14. DEFINITION. A substitution σ is a *unifier* of formulas ϕ, ϕ' on a branch B of a tableau T if it is the restriction of a substitution τ with the property $(\phi\pi)\tau = (\phi'\pi')\tau$ to $\text{Var} \setminus U$, where $U = \text{UVar}(\phi, B) \cap \text{UVar}(\phi', B)$ and π, π' are renamings of the variables in U with variables new to T .

With the closure rule based on this modified concept of unification, a tableau proof with less applications of expansion rules than in the standard calculus of tableaux with unification may be found; the calculus is strengthened.

Recognizing universal formulas is undecidable in general, however, a practically important subclass can be recognized easily (and this can already shorten tableau

⁵When obvious, a formula ϕ being universal on a branch B with respect to a variable x is just referred to as “the universal formula ϕ ,” and x as “the universal variable x .”

proofs drastically): in any sequence of tableau rule applications with a variable x introduced by a γ -rule application and not distributed over different branches by β -rule applications, all formulas generated during this sequence of rule applications are universal with respect to x , formally:

3.15. LEMMA. *A formula ϕ on a branch B of a tableau T is universal with respect to x on B if in the construction of T the formula ϕ was added to B by applying*

1. *a γ -rule and x is the free variable introduced;*
2. *an α -, γ -, or δ -rule to a formula that is universal on B with respect to x ; or*
3. *a β -rule to a formula β that is universal on B with respect to x , and x does not occur in any $\beta_i \neq \beta$.*

The proof of soundness of tableaux with unification (Theorem 3.12) can accommodate the universal formula technique. Bibel [1982] proposed a technique for reducing the size of proofs in the connection method, called *splitting by need*; like universal formulas it is based on the idea to avoid copying a universally quantified formula in cases where it is sound to use a single copy with different variable instantiations.

3.2.6. Binary versus n -ary Rules

The n -ary branching tableau rules for type β formulas in Table 2 have a binary variant

$$\frac{\beta}{\beta_i \mid \beta_1 \vee \cdots \vee \beta_{i-1} \vee \beta_{i+1} \vee \cdots \vee \beta_n} \quad (3.1)$$

which in fact is the more popular one and used, for example, in [Smullyan 1995, Fitting 1996]. Only recently, Massacci [1998a] pointed out that tableaux based on the n -ary rule cannot polynomially simulate tableaux based on (3.1) with respect to the minimal proof size, see also Section 6. In the present paper I work with the n -ary rule to achieve maximum uniformity among clausal and non-clausal tableaux. There is no loss of generality in doing so, because it is obvious that rule (3.1) can polynomially simulate the n -ary rule.

3.3. From Calculus to Proof Procedure

Tableau soundness gives the desirable property of tableaux with unification that a closed tableau for Φ signifies validity of $\bigvee_{\psi \in \Phi} \bar{\psi}$. There remains the question, whether for *all* valid sentences a tableau proof exists and, if this is the case, how it can be found. While the first question can be answered affirmative, for the second, a fully satisfactory answer is not yet available. This requires some explanation.

Definition 3.3 consists of a bunch of rules that define how to construct a tableau. In Section 3.4 it is shown that there exists a closed tableau with unification for

any given unsatisfiable set of sentences. This property of a calculus is called *completeness*. There is only a finite number of rules that can be applied to each given tableau, so it is a routine task to breadth first search for tableau proofs.

It would be much better, of course, if there were no need for search. Define a *tableau proof procedure* to be a tableau calculus equipped with a function F that, given a set of sentences Φ and a tableau T , computes in deterministic polynomial time (in the size of Φ and T) the next rule to be applied on T . It can be thought of as a “deterministic calculus”: its rules allow to construct at most one successor tableau from any given tableau and set of sentences. If the tableau proof procedure computes a tableau proof for any given unsatisfiable sentence, it is called *strongly complete*. The function F is called a *computation rule*. Let me point out why it is a difficult problem, to find strongly complete tableau proof procedures.

Usually, a great number of rules is applicable to any given tableau. More precisely, one must first select a branch B , where a rule is applied, then decide whether an expansion rule or a closure rule is used; in the first case one must choose a formula $\psi \in B \cup \Phi$, in the second case a pair of literals on B . Let us refer to these kinds of nondeterminism with the phrases *select branch*, *select mode*, *select formula*, and *select pair* in the following. In the propositional case no substitutions occur and, to arrive at a strongly complete tableau proof procedure, it suffices to select each non-literal formula exactly once on each branch in any order.

In the first-order case, one needs to apply rules more than once to certain formulas (otherwise, first-order logic were decidable). Making an arbitrary choice for a computation rule in the first-order case, however, results in general in an incomplete proof procedure.

In Figure 4, for example, the γ -formula is always preferred for expansion rule application, delaying expansion of the inconsistent propositional formula indefinitely. In an obvious way, the formula $Q \wedge \neg Q$ is treated *unfair*. This motivates the following definition.

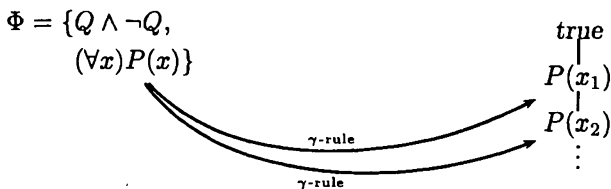


Figure 4: Incompleteness caused by unfair *select formula*.

3.16. DEFINITION. The set of tableaux with unification for a given set of sentences $\Phi \subset \mathcal{L}_\Sigma$ is partially ordered relative to a computation rule F , where the successor of a tableau T for Φ is the tableau computed from T by F . This defines a (possibly infinite, if Φ contains at least one type γ formula) ascending chain starting with the initial tableau T_0 for Φ and supremum T_∞ (which exists by Zorn's lemma).

A computation rule F is *fair* if for all Φ the following holds for all branches B in tableau T_∞ for Φ :

1. All formulas of type α , β , and δ occurring on B or in Φ were used to expand B (by applying the appropriate expansion rule)
2. All type γ formulas occurring on B or in Φ were used infinitely often to expand B (by applying the γ -rule).

It is simple to construct a fair computation rule, but this is not sufficient for strong completeness, because the above notion of fairness says nothing about closure. Combining fair application of the expansion rules with fair application of the closure rule, however, is a difficult problem, because tableaux with unification are *destructive*:

3.17. DEFINITION. A tableau calculus is *non-destructive* if all tableaux that can be constructed with the help of its rules from a given tableau T contain T as an initial subtree; otherwise the calculus is *destructive*.

For example, at first sight it might seem to be a good idea to apply the closure rule in a “greedy” manner, that is, as early as possible. Alas, it is not so. One problem is that several pairs of closure literals (with incompatible MGUs) may compete, but this is not all. In Figure 5, independently from which branch is closed first, the variable x_1 gets “used up” by a substitution that blocks closure of the other branch. Of course, a second free variable instance of the γ -formula may be created, but then the same happens one level below etc.

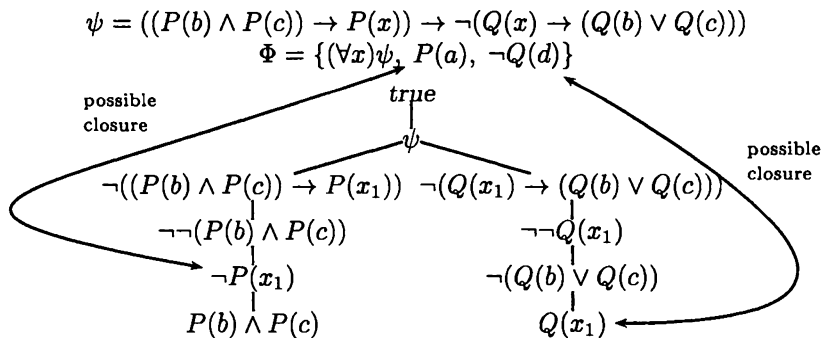


Figure 5: Incompleteness caused by unfair *select mode*.

Tableau with unification are (trivially) not destructive for propositional logic and for quantifier-free sets of sentences.

A non-destructive tableau calculus equipped with a fair computation rule gives a strongly complete proof procedure. Examples of non-destructive tableau calculi are Smullyan's [1995] ground tableaux and Fitting's [1996] tableaux with delayed instantiation rule, see below. As mentioned already, tableaux with unification *are* destructive. The culprit is the closure rule, Definition 3.3(3).

Independently of being destructive, a complete tableau calculus may fail to be proof confluent:

3.18. DEFINITION. A tableau calculus is *proof confluent*, if from every tableau for an unsatisfiable set of sentences a closed tableau can be constructed.

In other words, the search space of a proof confluent tableau calculus contains no “dead ends”, from where no proof can be found. A strongly complete tableau proof procedure is trivially a proof confluent tableau calculus. Thus, proof confluence is a necessary prerequisite for strong completeness.

A destructive tableau proof procedure still might be strongly complete, but as witnessed by the example in Figure 5, it might as well be not. At the present time, no strongly complete, destructive tableau proof procedure is known that works well in practice (there is hope, however, see Section 4.7). Therefore, it is worth discussing possible ways around the problem. Another reason is that some of the techniques dealing with destructiveness also deal with lack of proof confluence: some of the more important complete refinements of the tableau calculus are not proof confluent. This is discussed in Section 4.3 below.

An obvious way to tableaux with unification into a strongly complete proof procedure is to separate application of expansion and closure rules. Under a fair computation rule, delay the application of the closure rule until all tableau branches can be closed simultaneously by a suitable substitution. This is the path chosen in Fitting’s [1996] text book—and it has its inefficiencies: first, one cannot discard closed branches until the proof is essentially finished which might lead to storage problems (this can be partially remedied, see Section 3.5), and second, after each expansion rule application, the whole tableau must be tested for closure, which is very redundant. If more sophisticated data structures were used and the different MGUs available to close each branch were maintained as a tableau-wide constraint system that can be incrementally tested, then this approach might still be worth a try. There is experimental evidence to support this [Giese 2000].

Another option for implementing tableau proof search, which was mentioned already, comes from the observation that its nondeterminism is locally finite—from each tableau only a finite number of successor tableaux can be constructed. Envisage tableau proof search as a, possibly infinite, search tree whose nodes are tableaux. The root node contains the trivial tableau. The successors of a node are all the tableaux that can be constructed from it with one of the available tableau rules. Nodes that contain a closed tableau are success nodes. Even though the whole search tree is infinite, success nodes occur at finite depth and can be searched for in a breadth first manner. This approach is impractical, however, because of space requirements.

Stickel [1992] suggested to replace breadth first search by *depth first* search with backtracking and *iterative deepening* of the search depth (DFID search), which has only a small overhead in run time as compared to breadth first search, but is much more space efficient [Korf 1985].

DFID tableau proof search is based on a mapping m from \mathbb{N} to subsets of the tableaux that can be constructed with a given calculus, such that $\bigcup_{i \in \mathbb{N}} m(i)$ contains all these tableaux. Common choices⁶ for m , which is called *completion mode*,

⁶These options are implemented, for example, in the provers \mathcal{ZAP} [Beckert, Hähnle, Oel and

include the following:

- $m(i)$ = all tableaux with depth i
- ... with i nodes
- ... with i applications of γ -rule (per branch)
- ... with nesting depth i of terms

Now the parts of the search space containing the tableaux in $m(1), m(2), \dots$ are successively enumerated by depth first search with backtracking. To increase efficiency of the search it is important to get rid of as many nodes in the search space as possible. In the DFID setup this means to minimize the amount of backtracking. It is obvious that one may choose any deterministic strategy for *select branch* as all branches need to be closed eventually. In addition, it is not difficult to implement a fair computation rule that gets rid of *select formula* [Fitting 1996]. This leaves the—destructive—branch closure.

Tableau proof search based on DFID with backtracking over nodes corresponding to *select mode* and *select pair* is elegant and fast, when implemented in logic programming languages [Stickel 1992, Baumgartner and Furbach 1994, Beckert and Posegga 1995].

3.4. Tableau Completeness

The preceding discussion shows that it is difficult to ensure strong completeness of tableaux with unification. On the other hand, it is not too difficult to show mere existence of a closed tableau for each unsatisfiable set of sentences. The presentation of the latter result closely follows [Beckert and Hähnle 1998].

It is convenient to work with a data structure that slightly abstracts from tableau branches: so-called Hintikka sets (named after their inventor Hintikka) may contain an infinite number of formulas whose order is irrelevant. A model can be immediately constructed for any Hintikka set.

3.19. DEFINITION. A set $H \subset \mathcal{L}_{\Sigma^*}$ of sentences is a *Hintikka set* if it satisfies the following conditions:

1. $false \notin H$ and there are no complementary literals in H ;
2. if $\alpha \in H$, then all α_i are in H ;
3. if $\beta \in H$, then some β_i is in H ;
4. if $\gamma(x) \in H$, then $\gamma_1(t) \in H$ for all $t \in \mathcal{T}_{\Sigma^*}^0$;
5. if $\delta(x) \in H$, then $\delta_1(t) \in H$ for some $t \in \mathcal{T}_{\Sigma^*}^0$.

3.20. LEMMA (Hintikka). *Every Hintikka set is satisfiable.*

PROOF. An Herbrand model over the signature Σ^* is simply defined by setting $P^I(t_1, \dots, t_k) = true$ iff $P(t_1, \dots, t_k) \in H$ for $P(t_1, \dots, t_k) \in \mathcal{A}_{\Sigma^*}^0$. By induction on the structure of formulas in H it is easy to prove that $\mathbf{M} \models H$. \square

3.21. THEOREM (Completeness). *If the set $\Phi \subset \mathcal{L}_\Sigma$ of sentences is unsatisfiable, then there is a tableau proof for Φ .*

PROOF. Let $(T_n)_{n>0}$ be a sequence of tableaux for Φ constructed with a fair computation rule, containing no closure rule applications, and with limit T_∞ . We define a particular grounding substitution σ_∞ as follows: let $(B_k)_{k>0}$ be an enumeration of the branches of T_∞ and $(\phi_i)_{i>0}$ an enumeration of the γ -formulas in T_∞ . For every γ -formula ϕ_i , if ϕ_i occurs on B_k let x_{ijk} name the new variable introduced by the j -th application of the γ -rule to ϕ_i on B_k . (Note that different x_{ijk} can name the same variable.) Finally, let $(t_j)_{j>0}$ be an enumeration of \mathcal{T}_Σ^0 .

If we want to extract a model of Φ from B_k , then the instances of the ϕ_i on $B_k\sigma_\infty$ must “cover” all ground terms t_j . It suffices to choose $\sigma_\infty(x_{ijk}) = t_j$ for all $i, j, k > 0$. (If x_{ijk} and $x_{i'j'k'}$ name the same variable, then $i = i'$ and $j = j'$, so σ_∞ is well-defined.)

By construction of σ_∞ and fairness, if B is a branch in T_∞ and $B\sigma_\infty$ is open, then $B\sigma_\infty \cup \Phi$ is a Hintikka set and so Φ is satisfiable. This contradicts the assumption, hence $T_\infty\sigma_\infty$ is closed. The tree $T_\infty\sigma_\infty$ is finitely branching and the distance of all formulas involved in closures to the root node is finite. Then, by König’s Lemma⁷, there is an $n > 0$ such that the finite tableau $T_n\sigma_\infty$ is closed.

In general, σ_∞ is not a *most general* unifier of complementary literals used in closures and cannot be used in an MGU closure rule application to T_n . Therefore, it remains to show that σ_∞ can be suitably decomposed. This is done with a standard lifting argument: $\sigma_\infty = \sigma \circ \sigma_r \circ \sigma_{r-1} \circ \cdots \circ \sigma_1$, where σ_i is a most general closing substitution for the instance $B_i\sigma_1\sigma_2 \dots \sigma_{i-1}$ of the i -th branch in T_n ($0 < i < r+1$); σ is the part of σ_∞ not actually needed to close T_n . The σ_i are constructed inductively:

Let $\sigma'_1 = \sigma_\infty$. For $1 < i < r+1$, let σ_i be a most general substitution such that (1) σ'_{i-1} is a specialization of σ_i (there is a substitution σ'_i such that $\sigma'_{i-1} = \sigma'_i \circ \sigma_i$) and (2) σ_i is a closing substitution for $B_i\sigma_1\sigma_2 \dots \sigma_{i-1}$. Now σ_i is a most general *closing* substitution of $B_i\sigma_1\sigma_2 \dots \sigma_{i-1}$. Otherwise, there is a closing substitution σ''_i being more general than σ_i . The is-more-general relation is transitive, hence σ''_i is more general than σ'_{i-1} in contradiction to σ_i being already a suitable most general substitution. Finally, let $\sigma = \sigma'_r$. \square

It suffices to apply the appropriate expansion rule exactly once to each α , β , or δ -formula on each branch to obtain a Hintikka set from a fairly constructed sequence of tableaux. This has the practically relevant consequence that only to γ -formulas must a rule be applied more than once per branch.

3.5. Proof Representation

3.5.1. Trees, Matrices, Connections & Matings

Trees are quite a redundant way to represent proofs. Notably, each expansion step gives rise to new copies of some subformulas. This is unnecessary, as the result of an

⁷“A tree that is finitely branching but infinite must have an infinite branch.” A proof is, for example, in [Fitting 1996].

expansion step is uniquely determined once the position, where it is to be applied, is fixed. In the case of formulas in negation normal form (NNF), the situation is even simpler: up to variable instantiation, any formula occurring in a tableau for a formula ψ in NNF is just a subformula of ψ . Therefore, in the case of quantifier-free sentences in NNF, a tableau branch can be viewed as a sequence of positions of certain subformulas.

Such representations were first suggested independently by Davydov [1973], Bibel and Schreiber [1975], and Andrews [1976]. Bibel [1979] and Andrews [1981] defined procedures to check the validity of first-order formulas in NNF (in this case, one must record substitutions as well).

Full accounts of Bibel's *matrix* or *connection method* are [Bibel 1982b, Bibel 1987], of Andrews' *general matings* it is [Andrews 1981].

For the present discussion it is sufficient to define a *matrix* simply as an NNF formula not containing true, false, written in a two-dimensional notation: the immediate subformulas of a disjunctive formula are stacked vertically onto each other, while the immediate subformulas of a conjunctive formula are written in a horizontal row and enclosed between square brackets; literals are unchanged. To simplify things we start with propositional logic.

3.22. EXAMPLE. The NNF formula $\psi = P \wedge (\neg P \vee ((\neg P \vee Q) \wedge \neg Q))$ is represented as follows, where different occurrences of the same literal are distinguished by a superscript:

$$\left[\begin{array}{c} P \\ \left[\begin{array}{cc} \neg P^1 & \neg Q \\ Q & \\ & \neg P^2 \end{array} \right] \end{array} \right]$$

A *path* through a matrix M is a set $\pi = \{M_1, \dots, M_r\}$ of occurrences of submatrices (which can be literals) defined inductively:

1. For every matrix M , $\{M\}$ is a path through M (note that M can be a literal).
2. If M consists of rows M_1, \dots, M_r and π is a path through some M_i for $i = 1, \dots, r$, then π is a path through M .
3. If M consists of columns M_1, \dots, M_r and π_i is a path through M_i for all $i = 1, \dots, r$, then $\pi_1 \cup \dots \cup \pi_r$ is a path through M .

Some paths in the example are $\pi_1 = \{P, \left[\begin{array}{cc} \neg P^1 & \neg Q \\ Q & \end{array} \right]\}$, $\pi_2 = \{P, Q, \neg Q\}$,

while $\pi_3 = \{P, \neg P^1, \neg P^2\}$ is not a path.

Consider any branch of B of any non-trivial tableau for a propositional NNF formula ψ . Let π be those formulas of B that were not used as a premise of a rule application on B . Then it is fairly easy to prove by induction that π is a path through the matrix of ψ and, vice versa, each path through ψ is contained in a branch of some tableau for ψ . Thus, complementary formulas on branches are just complementary submatrices in paths. A pair of complementary formulas on a branch is called *connection* by Bibel and *mated* by Andrews.

In the NNF case, only connections between literals are possible. The paths of the example with only literals in them are $\pi_2, \pi_4 = \{P, \neg P^1, \neg Q\}$, $\pi_5 = \{P, \neg P^2\}$. One notices that each path contains a connection. As ψ is unsatisfiable, by soundness and completeness of tableaux and the correspondence between paths and tableau branches just stated, this is to be expected, of course. A set of connections \mathbf{C} such that each path through a matrix M contains a connection from \mathbf{C} is called *spanning* by Bibel [1981] who was the first to give this kind of matrix characterization of unsatisfiable formulas (without making use of the mentioned correspondence between matrices and tableaux):

3.23. THEOREM ([Bibel 1981]). *A propositional NNF formula ψ is unsatisfiable iff there is a spanning set of connections for its matrix.*

Bibel's [1982b] connection method and Andrews's [1981] general matings consist of a formal notation for matrices, paths, and connections together with a systematic procedure to find a spanning set of connections. It turns out that the paths these procedures look at correspond to the branches successively generated by certain tableau procedures.

An exact tableau counterpart to the non-clausal connection method with some additional restrictions is discussed in [Hähnle and Klingenbeck 1996]. The restriction of the connection method to clausal input corresponds exactly to weak connection tableaux with left-first branch selection discussed in Section 4.3.2 below.

Matrix methods were extended to first-order logic [Andrews 1981, Bibel 1982b]. In this case matrices contain additional notation to signify the kind and scope of quantifiers. True to the spirit of matrix methods, Skolem functions are avoided in [Bibel 1982b]. Instead, existentially quantified variables are considered as parameters that cannot be instantiated. To ensure soundness, ordering constraints on terms of the form " t may not occur as a subterm of t' " are being generated from the nesting structure of quantifiers. These constraints must then be satisfied by substitutions. The technique is independent of proof representation issues and, in fact, was employed for tableau calculi as well [Reeves 1987]. Its main advantage is that it generalizes to logics not permitting skolemization, such as intuitionistic logic [Voronkov 1996].

Just as γ -formulas need to be applied several times in tableau proofs, the scope of universally quantified submatrices must be present in a sufficient number of new instances, which is closely related to the Herbrand complexity in Theorem 2.3. Bibel [1982b] stresses that most of the structure of a universally quantified submatrix can be shared in an implementation. Again, the problems of proof search in destructive first-order calculi discussed in Section 3.3 are orthogonal to proof representation. Therefore, in practice, matrix methods tend to be implemented by DFID search [Bibel, Brüning, Egly, Korn and Rath 1995], just as tableaux with unification.

Matrix methods are closer to data structures allowing efficient implementation than tableaux. This positive feature, on the other hand, makes their the formal presentation of matrices very technical. I suspect that this a main reason why many refinements were conceived within the more abstract—and redundant—tableau for-

malism. Even worse, the less redundant structure of matrices can actually be in the way of extensions or optimizations: for example, certain rules needed to deal efficiently with formulas that contain equalities, add a new literal to a branch that is a logical consequence of a literal set C on the same branch (for example, the basic superposition calculus of [Degtyarev and Voronkov 1998])—this cannot be done in an obvious way within a matrix framework, because the paths containing C are not explicitly represented. Similarly, simplification as discussed in Section 3.5.3 below, cannot be easily incorporated into a matrix framework.

But there is an important merit of matrix formulations compared to tableau methods, besides taking implementation issues seriously: for instance, there are sound transformations on the matrix level, called *reduction* in [Bibel 1982b], that cannot necessarily be efficiently simulated on the level of paths or branches. More generally, the global view suggested by matrices may very well lead to refinements difficult to detect with the path- or branch-based view of tableaux. Evidence of this consideration is provided by Letz [1998] who defined a tableau refinement based on the observation that one spanning *set* of connections gives possibly rise to many different tableaux that differ only in the sequence in which these connections occur on the branches.

An extensive overview over various calculi from the point of view of matrices is [Bibel and Eder 1992].

Finally, it should be mentioned that apart from matrices further formula representations exist that try to avoid redundancy: I want to mention clausal [Gallo and Urbani 1989] and non-clausal [Preiß 1998] *hypergraphs* and binary decision diagrams (BDD) [Bryant 1986]. While hypergraphs are an alternative notation for formulas and can be computed in linear time, BDDs combine normal form computation and deduction, in fact, a BDD is a normal form of a propositional formula from which its models can be directly read off. Both, hypergraphs and BDDs are closely related to tableaux [Posegga 1993, Preiß 1998]. They share, however, the drawback that their generalization to first-order logic so far has proven to be problematic [Rago 1994, Posegga and Schmitt 1995].

3.5.2. *Pruning Irrelevant Parts of a Proof*

Pruning, which is closely related to the *condensing* technique of Oppacher and Suen [1988], allows the reduction of both the size of the search space and the size of generated tableau proofs. It appears in the literature also under the name *level cut* [Baumgartner, Furbach and Niemelä 1996]. Koshimura and Hasegawa [1999] showed that condensing is a special case of the *non-Horn magic set* transformation [Hasegawa, Inoue, Ohta and Koshimura 1997] which in turn was shown [Ohta, Inoue and Hasegawa 1998] to be essentially the same as *relevancy testing* [Loveland, Reed and Wilson 1995].

Suppose a branch B of a tableau was expanded by a β -rule application and one of the extensions β_i was *not* used to close the subtableau T_i below β_i , then T_i is still closed when appended to any of the other extensions β_j , $j \neq i$, or even when appended immediately below B (define an extension β_i to be *used*, if β_i itself or

any of the formulas resulting from it through tableau rule application is used in an application of the closure rule). To take advantage of this situation, either the closure rule is changed such that all branches in the tableau containing B as a subbranch are considered to be closed, or—similarly—all branches containing one of the β_j are *pruned*, that is, the effects of the β -rule application are undone, see Figure 6.

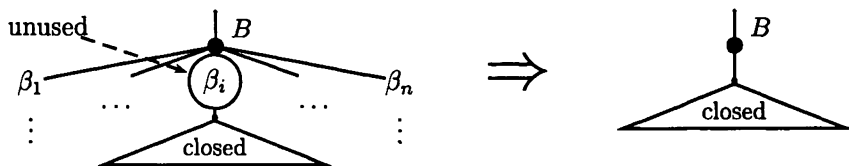


Figure 6: Pruning irrelevant parts of a tableau proof.

3.5.3. Simplification

The benefits of intermediate simplification steps to be applied after each tableau rule application is stressed by Massacci [1998b]. The idea is that for each propositional formula ψ present on a branch B each positive occurrence of ψ as a subformula can soundly be replaced by *true* while each negative occurrence can be replaced by *false* with subsequent simplification steps of the form $\text{true} \vee \theta \Rightarrow \text{true}$, etc. In contrast to branching, simplification is an inexpensive operation and can be computed in (low) polynomial time in the size of formulas on branches. The well-known unit resolution and pure literal rule subprocedures of the Davis-Putnam-Loveland-Logeman procedure [Davis, Logemann and Loveland 1962] (see also Section 5.4) are special cases of Massacci's [1998b] simplification rule who demonstrated its effectiveness for (modal) propositional logic. It remains to be seen, however, if a useful variant for first-order tableau with unification will emerge.

4. Clause Tableaux

In the present section a number of refinements of the tableau procedure are introduced. For several reasons, these refinements are discussed on the clause level:

- Simplified notation leads to easier detection of new refinements
- Efficient implementability, for example, by compilation to abstract machines
- Completeness proofs stay manageable
- Comparability (most deduction procedures are implemented on the clause level)

Restricting attention to the clause level implies some limitations as well:

- Some applications (such as software verification) expect proofs on the non-clausal level: back-translation from clauses can be tricky
- For some non-classical logics a clause normal form is unknown
- Proofs become harder to read for humans

- Some applications (such as computing prime implicants) require the models of a formula to be preserved, and then in the worst case its CNF has exponential size

In summary, there is considerable incentive to generalize the results that follow (partially this has been done, for example, in [Hähnle and Klingenberg 1996, Hähnle, Murray and Rosenthal 1997]), but I believe the material to be more accessible in the present, syntactically limited form.

4.1. Normal Form Computation

How first-order sentences are efficiently transformed into sets of clauses is shown, for example, in [Plaisted and Greenbaum 1986, Nonnengart, Rock and Weidenbach 1998], and in [Baaz et al. 2001, Nonnengart and Weidenbach 2001] (Chapters 5 and 6 of this Handbook).

4.2. Clause Tableau Proofs, Soundness, Completeness

4.2.1. Clause Tableaux

Let us start by stating suitably simplified versions of Definitions 3.3 and 3.4.

4.1. DEFINITION. Given a set S of clauses from \mathcal{L}_Σ , a *clause tableau for S* is defined as a tableau constructed with the following rules:

- (i) The tree consisting of a single node labeled with *true* is a tableau for S (initialization rule).
- (ii) Let T be a tableau for S , B a branch of T , and $L_1 \vee \dots \vee L_r$ a new instance of $C \in S$. If the tree T' is constructed by extending B with r new subtrees and the nodes of the new subtrees are labeled with L_i , then T' is a tableau for S (extension rule).
- (iii) Let T be a tableau for S , B a branch of T , and L and L' literals on B . If L and $\overline{L'}$ are unifiable with MGU σ , and T' is constructed by applying σ to all literals in T (that is, $T' = T\sigma$), then T' is a tableau for S and branch $B\sigma$ is marked as *closed* (closure rule).

Clauses are first-order formulas, so the extension rule is composed of several applications of the expansion rule 3.3(2), which justifies the change in terminology.

4.2. DEFINITION. A clause tableau T for a set S is *closed* if *all* its branches are marked as closed.

A *clause tableau proof* for (the unsatisfiability of) a clause set $S \subset \mathcal{L}_\Sigma$ is a closed clause tableau T for S .

4.3. EXAMPLE. The formula set of Example 3.6 can be transformed in a CNF that consists of the following clauses:

- (1) $P(c)$
- (2) $\neg P(x) \vee Q(x)$
- (3) $\neg Q(x) \vee R(x)$
- (4) $\neg P(x) \vee \neg R(x)$

A clause tableau for this clause set is displayed in Figure 7. Observe that the nodes are a subset of the nodes of the tableau in Figure 2.

In the following, closed branches are not indicated by arrows between participating literals anymore, but merely by a horizontal bar and the closing MGU below their leaf.

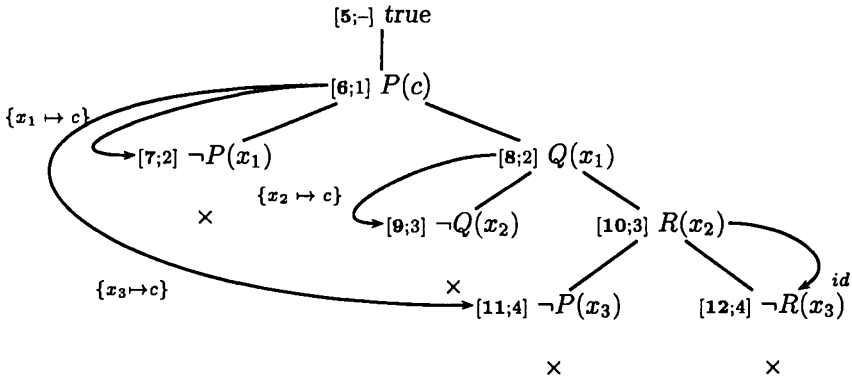


Figure 7: Clause tableau proof of Example 4.3.

Clause tableaux mainly constitute a syntactic simplification of full first-order tableaux. The main properties of the calculus are the same, in particular the discussion in Section 3.3 applies to them as well.

In contrast to the full first-order case the extension and closure rule only use clauses from the input set and branch literals. This simplifies some definitions.

4.2.2. Soundness and Completeness

Soundness of clause tableaux follows immediately from Theorem 3.12 by observing that clauses are particular first-order formulas and extension rule 4.1(ii) can be composed of several applications of rule 3.3(2).

Completeness could be obtained easily by suitable simplification of the proof of Theorem 3.21, but in the clausal case a more modular approach is useful. Following Robinson [1965], lifting a ground proof to a first-order proof is separated from proving ground completeness of a calculus. The advantage is that the lifting part is similar for all completeness proofs of the following tableau refinements and either

is obvious or at most requires a sketch. So it is sufficient to concentrate on ground completeness. Abstraction from first-order issues greatly simplifies completeness proofs of the more complicated calculi that follow.

4.4. THEOREM (Lifting). *Let S be a clause set, \hat{S} a set of ground instances of S and \hat{T} a clause tableau proof for \hat{S} . Then there is a clause tableau proof T for S and a substitution τ such that $\hat{T} = T\tau$.*

PROOF. The main technical difficulty of this proof is that in Definition 4.1(iii) only MGUs are to be used whereas \hat{S} may contain arbitrary ground instances of clauses. The following property of MGUs is needed:

If T' is a clause tableau, τ a substitution such that $T'\tau$ is closed, and ρ an MGU that closes any branch of T' , then $T'\rho\tau = T'\tau$. (4.1)

Proof of (4.1): by definition of an MGU and as τ closes T' , there is τ' with $\rho\tau' = \tau$. MGUs can be assumed to be idempotent, so $T'\rho\tau = T'\rho\rho\tau' = T'\rho\tau' = T'\tau$.

Back to the main proof, let T^0 be constructed exactly as \hat{T} but for each extension step with $\hat{C} \in \hat{S}$ used in \hat{T} , take instead a new instance of the clause $C \in S$ of which \hat{C} is a ground instance. Obviously, $T^0\tau = \hat{T}$ for a suitable grounding substitution τ .

If B is an arbitrary open branch of T^0 , then it is closed by τ , so there is an MGU ρ that closes B and rule 4.1(iii) is applicable to obtain a clause tableau $T^1 = T^0\rho$. By (4.1), $T^1\tau = T^0\tau = \hat{T}$. Repeating this argument in a straightforward induction over the number n of open branches in T^0 yields a clause tableau $T = T^n$ such that $T\tau = \hat{T}$. \square

In the proof the sequence of branch closures was arbitrary which shows independence of the *select branch* strategy.⁸

As to completeness, let us look first at the ground case. To minimize iterated efforts, we proceed in a schematic way. The following *ground completeness schema* for any given clause tableau restriction, let us call it X-tableau, is proven:

If the finite ground clause set S is unsatisfiable, then there is an X-tableau proof for S . (4.2)

We could proceed to prove ground completeness of unrestricted clause tableaux right now, but in following sections ground completeness of various restrictions of clause tableaux is proven, of which completeness of the unrestricted calculus is an immediate consequence.

4.5. PRINCIPLE (Schematic Completeness). If the clause set S is unsatisfiable, then there is an X-tableau proof for S .

⁸When the computation rule of *select clause* is arbitrary, but fair, the theorem still holds in the weakened form that there is a clause tableau T for S such that $T\tau$ appears as a subset of the nodes of \hat{T} .

PROOF (*Schema*). Herbrand's Theorem 2.3 provides a finite, unsatisfiable set \hat{S} of ground instances of S . By a suitable instance of (4.2) there is a closed ground X-tableau \hat{T} for \hat{S} and, by Theorem 4.4, there is a closed X-tableau for S , whenever X has the lifting property: if $T\tau$ is an X-tableau, then T is an X-tableau as well. \square

As announced already, the next goal is to find complete restrictions of clause tableaux. It is sufficient to prove a suitable instance of (4.2), whenever a ground X-tableau proof \hat{T} lifts to a first-order X-tableau proof T . It is usually sufficient to check that the proof of Theorem 4.4 can be used unaltered.

From the point of view of proof search, restricting the tableau calculus means to exclude certain choices in *select clause* and *select pair* and to fix *select branch* in some way.

4.3. Connections

Connection conditions were pioneered by Andrews [1981] and Bibel [1982b].

4.3.1. Connection Tableaux

A major drawback of the tableau calculus is that the extension rule 4.1(ii) is applied completely unguided which can clutter up tableaux with many nodes that do not contribute to a proof.

4.6. EXAMPLE. Consider the two clause tableaux for $S = \{P(x) \vee Q(x), R(x) \vee S(x), \neg P(a), \neg Q(a), \neg R(b), \neg S(b)\}$ displayed in Figure 8. The tableau on the right constitutes a minimal proof, while the second extension step in the tableau on the left is completely unrelated to the initial step.

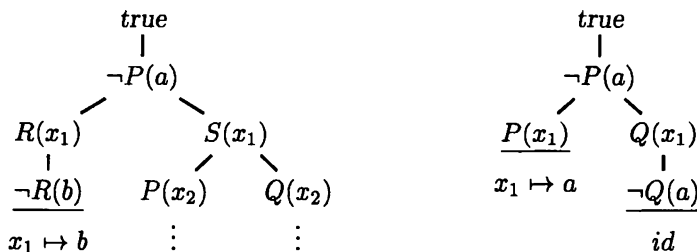


Figure 8: Redundant nodes in a tableau.

4.7. DEFINITION. A *connection tableau* is a clause tableau in which every inner node L (except $true$) has \bar{L} as one of its immediate successors [Letz, Schumann, Bayerl and Bibel 1992].

The tableau on the right in Figure 8 is a connection tableau, the tableau on the left is not. It is excluded by the connection restriction.

Connection tableaux are complete, but the proof is deferred until the next section. The definition of connection tableaux implies that when $T \neq \text{true}$ at least one of the new branches generated by the tableau extension rule can be closed. This suggests a *procedural* definition of connection tableaux obtained from Definition 4.1 by changing the first two rules:

- (i') For any new instance $L_1 \vee \dots \vee L_r$ of $C \in S$ the tree constructed by extending *true* with r new subtrees with nodes L_i is a connection tableau for S .
- (ii') Let T be a tableau for S , B a branch of T ending with L , $L_1 \vee \dots \vee L_r$ a new instance of $C \in S$. If \bar{L}, L_i (where $i \in \{1, \dots, r\}$) are unifiable with MGU σ and the tree T' is constructed by extending B with r new subtrees, where the nodes of the new subtrees are the L_j , then $T'\sigma$ is a connection tableau for S , in which the branch ending with $L_i\sigma$ is marked as closed. This is called a *connected extension step*.

Closure of open branches (Definition 4.1(iii)) is unchanged and called *reduction step*. Note that, besides in reduction steps, branches can be in addition be closed in extension steps. This justifies the change of terminology.

It is important to note that while clause tableaux are proof confluent, connection tableaux are not:

4.8. PROPOSITION. *Ground connection tableaux are not proof confluent.*

PROOF. Consider $S = \{P, \neg P, Q\}$ and let Q be the clause used in the initial step. It is impossible to make any further extension step, although S is clearly unsatisfiable. (Examples independent of the choice of the initial clause can be found in [Letz 1993].) \square

In Section 3.3 it was pointed out that a “greedy” strategy for preferring closure over extension steps leads to incompleteness. It is tempting to employ a greedy strategy at least for closures occurring within reduction steps, but the following counter example due to Letz⁹ shows that even this results in incompleteness:¹⁰

4.9. EXAMPLE. If $S = \{P(a) \vee P(x) \vee Q(x), \neg Q(b) \vee R, \neg P(b) \vee R, \neg R\}$, then $\{\neg P(a)\} \cup S$ is unsatisfiable.

A proof starting with $\neg P(a)$ must use $P(a) \vee P(x) \vee Q(x)$ in the first extension step. A left-first *select branch* rule leads to greedy reduction with $P(x)$ (and $\neg P(a)$) and the proof is stuck at the open branch containing $Q(a)$. With a different starting clause, a proof with greedy reduction is possible, but a trick taken from [Letz, Mayr and Goller 1994] gives a general counter example: let S' be as S , but P replaced with P' , Q with Q' and R with R' . Then $\{\neg P(a) \vee \neg P'(a)\} \cup S \cup S'$ is still unsatisfiable.

⁹Personal communication.

¹⁰In fact, the authors of SL-resolution [Kowalski and Kuehner 1971], discussed as a close relative to connection tableaux in Section 5.2, were tempted enough: they suggested reducing greedily without noticing the incompleteness problem.

Regardless of the starting clause, however, $\neg P(a) \vee \neg P'(a)$ must be used at some point. This cannot be the last extension step in a proof, because the signatures of S and S' are disjoint. Therefore, the proof gets stuck in the same way as above.

4.3.2. Weak Connections

The extension rule (ii') of connection tableaux, however, has a natural relaxation that partially restores proof confluence:

- (ii'') Let T be a tableau for S , B a branch of T containing L not necessarily as leaf, $L_1 \vee \dots \vee L_r$ a new instance of $C \in S$. If \bar{L} , L_i (where $i \in \{1, \dots, r\}$) are unifiable with MGU σ and the tree T' is constructed by extending B with r new subtrees, where the nodes of the new subtrees are the L_j , then $T'\sigma$ is a clause tableau for S , in which the branch ending with $L_i\sigma$ is marked as closed. (This is called a *weakly connected extension step*.)

Let us call the resulting calculus *weak connection tableaux*.

4.10. DEFINITION. A clause set is *minimally unsatisfiable (mu)* when it is unsatisfiable and each of its proper subsets is satisfiable. A clause is *relevant* in S when it is contained in a mu subset of S .

It can be shown that weak connection tableaux are proof confluent provided that *select clause* is implemented in a fair manner and the initial clause is relevant. Unfortunately, testing for membership in a mu set is as expensive as testing unsatisfiability itself. This limits the usefulness of weak connection tableaux in practice, but in Section 4.5 a slight relaxation is the basis of a whole class of interesting calculi which are proof confluent regardless of the initial clause.

4.4. Regularity

An important device in tableau-based theorem proving that avoids constructing certain redundant proofs is *regularity*:

4.11. DEFINITION. A clause tableau is *regular*, if none of its branches contains more than one occurrence of the same literal.

4.12. EXAMPLE. Regularity can help to avoid substitutions that lead to redundant proofs. Consider the tableau for $S = \{P(0), \neg P(x) \vee P(s(x)), \neg P(s(s(0)))\}$ in Figure 9. The first possible substitution for the middle branch renders the right branch irregular and is thus avoided.

Implementing regular tableaux is not straightforward, because an admissible closure substitution can potentially unify as well formerly different literals on branches closed already. For efficiency reasons one discards closed branches immediately, so there must be a mechanism to exclude such critical substitutions. It was suggested in [Letz et al. 1992] to create an inequality constraint of the form $t_1 \neq t'_1 \vee \dots \vee t_m \neq t'_m$,

whenever two unifiable literals $L(t_1, \dots, t_m)$ and $L(t'_1, \dots, t'_m)$ are encountered on one branch. Similar constraints are generated to characterize tautologous instances of clauses used in extension steps. Then substitutions are applied to constraints as well and must ensure their satisfiability. In the example above, the second extension step generates the constraint $s(0) \neq s(x_2)$ which is not satisfied by $x_2 \mapsto 0$.

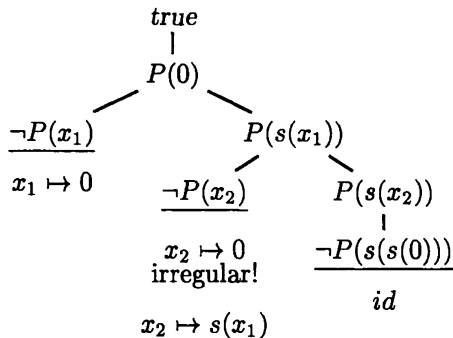


Figure 9: Advantage from regularity.

The following standard lemma is needed in the proof of ground completeness of regular connection tableaux. Its easy proof is given, for instance, in [Loveland 1978, Lemma 2.3.2, p. 63]. The completeness theorem below was first proven (differently) in [Letz 1993]. The present proof is from [Hähnle et al. 1997].

4.13. LEMMA. *Let S be a mu ground clause set with $C \in S$, $D \subseteq C$, and $S_D = (S - \{C\}) \cup \{D\}$. Then for any mu subset S'' of S_D : (i) $D \in S''$; (ii) $D \not\subseteq D''$ for all $D \neq D'' \in S''$.*

4.14. THEOREM (Completeness). *If the finite ground clause set S is unsatisfiable, then there is a regular connection tableau proof for S .*

PROOF. We show by induction on the number k of literal occurrences in S : for any relevant clause $C_j \in S$, that is not a unit clause, there is a closed regular connection tableau for S whose initial step uses C_j . If there is no such clause, there must be a mu subset of unit clauses in S ; it is trivial to find a regular connection tableau proof for such a set.

$k \in \{0, 1, 2\}$: either S is satisfiable or it contains only unit clauses or the empty clause and the claim is trivially satisfied.

$k > 2$: (see Figure 10) let $C_j = L_1 \vee \dots \vee L_i \vee \dots \vee L_n$ be a relevant non-unit clause in S and let T be the regular connection tableau consisting just of an initial step that uses C_j (upper middle part of Figure 10).

For all $i \in \{1, \dots, n\}$ let $C'_j = L_i$ and $S_{L_i} = (S - \{C_j\}) \cup \{C'_j\}$. By Lemma 4.13(i), C'_j is contained in an mu subset S'_{L_i} of S_{L_i} . Hence, \bar{L}_i occurs in a clause C^i of S'_{L_i} . Moreover, S'_{L_i} contains less literals than S .

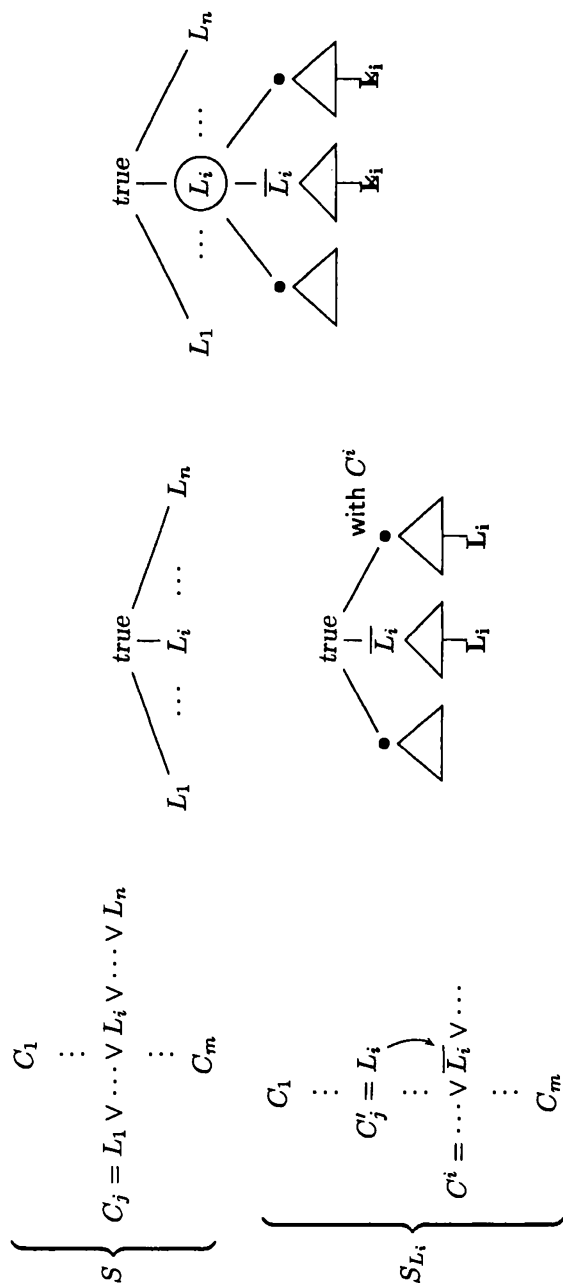


Figure 10: Illustration of the proof of Theorem 4.14

If C^i is a unit clause, then the i -th branch of T can be closed immediately, resulting in a regular connection tableau. Otherwise, applying the induction hypothesis on C^i and S'_{L_i} yields a closed regular connection tableau T_i for $S_{L_i} \supseteq S'_{L_i}$, where the first extension step uses C^i (lower middle part of Figure 10).

By Lemma 4.13(ii), L_i occurs at most in C'_j and is therefore only used in extension steps with the unit clause C'_j in T_i (highlighted by boldface type in the figure). As shown in the figure, each T_i is glued together with T at L_i maintaining connectedness. In the resulting tableau, irregularity can at most occur with the L_i . But as L_i occurs on top of each T_i (circled occurrence) the extension steps with unit clause L_i simply can be replaced by reduction steps with the circled occurrence of L_i . Because of $S_{L_i} - \{C'_j\} \subseteq S$, the result is a regular connection tableau for S . \square

The proof has an interesting consequence resulting in a restriction for initial clauses, which is of importance later on in Section 5.3: each mu set of (not necessarily ground) clauses S trivially contains a negative clause (if not, it can be satisfied by the constantly true interpretation). Hence, one of the negative clauses of S is relevant and, therefore, the initial extension step can be restricted to negative clauses.

4.15. COROLLARY. *Regular connection tableaux are complete even when the first extension step must use a negative clause.*

4.5. Orderings and Selection Functions

4.5.1. Redundancy and Saturation in Tableaux

Let us take up the theme expressed in the regularity restriction, namely to avoid redundancy in tableau proofs.

Any open branch B in a ground clause tableau T for S , or equivalently, any consistent set of ground literals B defines a partial *interpretation* I_B on S via $I_B \models L$ iff $L \in B$.

A first-order clause tableau is rendered irregular by an extension step of branch B with a non-tautologous clause C iff $I_{\exists B} \models C$, where $\exists B$ is obtained from B by replacing its variables with new and differing constant symbols. When B is ground, regularity, therefore, amounts to avoiding extension, whenever $I_B \models C$ holds. A stronger notion of redundancy is desirable, though. Until further notice we work with ground clauses.

4.16. DEFINITION. An open clause tableau branch B has a *saturation* with respect to a clause set S iff it has an extension $\tilde{B} \supseteq B$ such that $I_{\tilde{B}} \models S$.

It is, of course, not realistic to consider *all* possible extensions of a branch (the empty branch, for example, always has a saturation when S is satisfiable), so we check only one of them to guide tableau extension.

If I_B is not yet a model of S (that is, B is not a saturation itself), then there must be a reason for it in the form of clauses $C \in S$ not satisfied by I_B . We try to

complete I_B to a model of all clauses in S by adding to it selected literals from the unsatisfied clauses. These clauses are, by definition, no tautologies and are regular on B .

This idea is formalized in the following section.

4.5.2. Tableaux with Selection Function

Let f be a *selection function* on clauses: a function mapping each clause into a (possibly empty) subset of its literals. The *extension* \tilde{B}_f of B with respect to f is defined as follows:

$$\tilde{B}_f = B \cup \bigcup_{\substack{C \in S \\ I_B \not\models C}} f(C) \quad (4.3)$$

If \tilde{B}_f is consistent and all clauses with no selected literals were used on B , then $I_{\tilde{B}_f} \models S$ by construction and proof search can be stopped here. In general, however, \tilde{B}_f does not induce an interpretation, because it may contain complementary literals. In this case, one of the clauses not yet satisfied by I_B is selected for extension whose selected literal(s) contribute to a contradiction in \tilde{B}_f . Formally, let $\overline{f(C)}$ denote the set of complements of literals selected by f in C . Then extension steps (Definition 4.1(ii)) are restricted to clauses in

$$\{C \mid C \in S, I_B \not\models C \text{ and } (\overline{f(C)} \cap \tilde{B}_f \neq \emptyset \text{ or } f(C) = \emptyset)\} \quad (4.4)$$

By definition, if $L \in \overline{f(C)} \cap \tilde{B}_f$, either $L \in B$ or $L \in f(D)$ for some clause $D \in S$. In the first case the extension is a *weak connection step* in the sense of (ii'') on page 131 (accordingly, the branch containing $\bar{L} \in f(C)$ is marked as closed). The second case and the case when no literal is selected are called a *restart step* (and C a *restart clause*) to emphasize that this part of a tableau proof bears no direct connection with the current branch. The top clause in a tableau proof is always a restart step. No new restart clauses are added once a selection function f is fixed: they can be computed in advance.

4.17. EXAMPLE. Consider the clause set $S_1 = \{\neg Q \vee \neg \underline{S}, \neg R \vee \underline{S}, P \vee Q \vee \underline{R}, \neg \underline{P}\}$ in which the lexicographically largest literal is selected (these are underlined). Initially, $B = \{\text{true}\}$ and $\tilde{B}_f = \{\neg S, S, R, \neg P\}$. The first two clauses are the only restart clauses of which the first is selected (see Figure 11).

For the leftmost branch $B = \{\neg Q\}$ one obtains $\tilde{B}_f = B \cup \{S, R, \neg P\}$, which models S_1 . On the other branch $B = \{\neg S\}$ one has $\tilde{B}_f = B \cup \{S, R, \neg P\}$, so an extension step (the only one) with the second clause is possible. The only open branch is now $B = \{\neg S, \neg R\}$ with $\tilde{B}_f = B \cup \{R, \neg P\}$ and only extension with the third clause is allowed. The first open branch, $\{\neg S, \neg R, P\}$ is closed by a further extension while the last open branch $B = \{\neg S, \neg R, Q\}$ yields $\tilde{B}_f = B \cup \{\neg P\}$ and thus the second model of S . Observe that all but the very first extension step were

determined which demonstrates the potential for down-sizing the search space with selection functions.

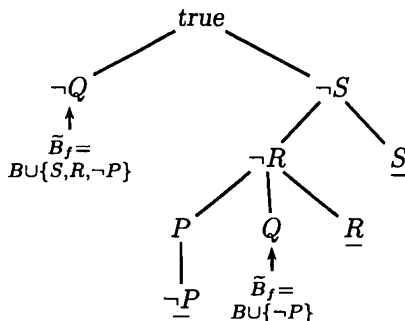


Figure 11: Tableau with selection function.

4.18. THEOREM. *If the finite ground clause set S is unsatisfiable, then for any selection function f there is a tableau proof with selection function f for S .*

PROOF. Assume there is a tableau with selection function f and an open branch B in which all possible extension steps were made. As S is finite, B is finite as well. We claim that \tilde{B}_f is consistent from which $I_{\tilde{B}_f} \models S$ follows by construction (in particular, all clauses with no selected literal are satisfied). If \tilde{B}_f is inconsistent there is a literal $L \in \{f(C) \mid C \in S, I_B \not\models C\}$ such that \bar{L} occurs (I) in B or (II) in $f(D)$ for some clause D not yet satisfied by B . In case (I) a weakly connected extension step is possible on B in case (II) a restart step is admissible. Either way, the assumption that all possible extension steps were made on B is contradicted. \square

The proof is independent of the sequence of extension steps chosen, so tableaux with selection function are *proof confluent*. Moreover, a slight generalization of the proof shows that completeness is retained even when f is changed during tableau construction.

4.5.3. Related Calculi

A number of recently suggested restrictions of clause tableaux can be considered as special cases of tableaux with selection function, for example, ordered tableaux [Klingenbeck and Hähnle 1994, Hähnle and Klingenbeck 1996].

4.19. DEFINITION. A ground *literal (L -)ordering* is a binary relation $<$ on ground literals which is irreflexive and transitive.

L-orderings give a complete tableau restriction which can be expressed via selection functions as follows: simply use

$$f_{<}(C) = \{L \mid L \text{ maximal in } C \text{ with respect to } <\}.$$

The restriction $I_B \not\models C$ in (4.3), (4.4) is not enforced by Hähnle and Pape [1997] and $|f(C)| > 0$ is required, otherwise their calculus is identical to the present version of tableaux with selection function. On the other hand, Pape and Hähnle [1997] showed that tableaux with selection functions can be modified to accommodate connected extension steps.

In ordered tableaux, by definition, $f(C) \neq \emptyset$; hence, only restart clauses of the kind that are connected via f can occur. It is a natural question, if one can get rid of restart clauses altogether. In [Hähnle and Klingenbeck 1996] it is shown that ordered tableaux without restart steps (that is *each* extension step is weakly connected via f) are incomplete for certain total orderings. It is not obvious, for which selection functions a calculus without restart clauses might be complete (besides the trivial selection function defined by $f(C) = C$ for all $C \in S$, which gives regular clause tableaux without any further restriction).

On the other hand, one can impose a restriction, which is complementary to that of ordered tableaux, in the sense that one permits *only* restart clauses of the kind, where $f(C) = \emptyset$:

4.20. DEFINITION. A selection function is *consistent* (with respect to S) if $\bigcup_{C \in S} f(C)$, the set of all literals in a clause set S selected by f , is consistent.

Whenever f is consistent with respect to S , only restart steps with clauses that have no selected literals are possible. The special case when S is consistent, $|f(C)| \leq 1$, and there is exactly one restart clause in S , is a complete calculus known in the literature as *SL-resolution without contrapositive clause variants* (*SLWV-resolution*) [Pereira, Caires and Alferes 1992].

4.5.4. First-Order Issues

Lifting is straightforward for tableaux with selection function f provided that f lifts. More precisely, call f *stable with respect to substitutions* if $f(C\sigma) \subseteq f(C)\sigma$ for all clauses C and substitutions σ . For L-orderings this translates into the requirement $L < L'$ implies $L\sigma < L'\sigma$ for all substitutions σ and literals L, L' . It is obvious from the discussion following 4.4 that tableaux with selection functions that are stable with respect to substitutions lift without problems.

Implementation of first-order tableaux with selection function poses similar problems as regularity. In addition to regularity constraints, selectedness constraints are derived from (4.4) and take the form $L \in f(C)$ [Pape 1996, Hähnle and Pape 1997].

Checking selectedness constraints for satisfaction can be expensive (NP-complete). If one decides to suppress their generation, then the resulting calculus can be called *tableaux with input selection function* [Hähnle and Pape 1997], because the selection restriction is only enforced on clauses that serve as input for extension steps, but

not on instances of clauses used in a tableau already. This has another advantage: it was noted after the proof of Theorem 4.18 that the selection function may be arbitrarily changed during tableau construction. This implies at once that selection functions need not be stable with respect to arbitrary substitutions in tableaux with input selection function, rather, stability with respect to variable renamings is sufficient [Hähnle and Pape 1997].

4.6. Hyper Tableaux

Recently, tableau calculi based on hyper extension rules gained considerable attention [Bry and Yahya 1996, Baumgartner et al. 1996, Shults 1997, Baumgartner 1998]. This is not surprising, because hyper-resolution [Robinson 1965a] is long known to be a key ingredient to success in theorem proving. Hyper calculi share the feature that several deduction steps are combined into one. This yields a speed-up in proof search, but the main advantage is that some intermediate results are not computed in the first place and this can limit the search space considerably. In fact, hyper tableaux were considered early on by Brown [1978], but this work did not make the impact it deserved. The family of calculi known as model generation [Manthey and Bry 1988, Fujita and Hasegawa 1991] is essentially a variant of hyper tableaux and is discussed below.

It is well-known that hyper-resolution can be seen as an instance of semantic resolution [Slagle 1967]. The same kind of generalization is done in the following for hyper tableaux.

Again, we start with the ground case. One stipulates a similar condition as (4.4) on clause candidates for extension saying that *all* selected literals of an extending clause must be weakly connected to the current branch, formally, each clause used in an extension step (Definition 4.1(ii)) on B must be from the set:

$$\{C \mid C \in S, \mathbf{I}_B \not\models C \text{ and } \overline{f(C)} \subseteq B\} \quad (4.5)$$

A ground clause tableau constructed with this restriction is called a *hyper tableau*. In each extension step the branches containing complements of selected literals of the extending clause are marked as closed.

Given a selection function f and a clause C with $f(C) = \{L_1, \dots, L_m\}$ and $C - f(C) = \{L_{m+1}, \dots, L_n\}$, one can rewrite C as a rule in the following fashion:

$$\overbrace{L_1 \wedge \dots \wedge L_m}^{\text{selected literals}} \rightarrow \overbrace{L_{m+1} \vee \dots \vee L_n}^{\text{not selected literals}} \quad (4.6)$$

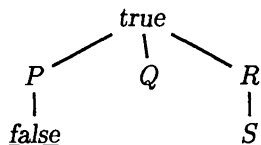
The premise of C viewed as a rule (that is, the set of literals $\{\overline{L_1}, \dots, \overline{L_m}\}$) is equivalent to $\{\text{true}\}$, when there are no selected literals; likewise, the conclusion of C viewed as a rule (that is, the set of not selected literals $\{L_{m+1}, \dots, L_n\}$) is equivalent to $\{\text{false}\}$, when there are only selected literals. Note that the premise of a rule contains *complements* of selected literals.

As *true* occurs on every tableau branch, the following reformulation of (4.5) is possible:

$$\{C \mid C \in S, \mathbf{I}_B \not\models C \text{ and all literals of the premise of } C \text{ occur in } B\} \quad (4.7)$$

Clauses with no selected literals are unrestricted in application and, therefore, play the rôle of *restart clauses* in hyper tableaux. In the following the convention is adopted not to display branches closed by extension steps and to treat *false* as a literal indicating closure by clauses having only selected literals.

4.21. EXAMPLE. Let f_N select exactly the negative literals of each clause. Ground clause set $S_2 = \{\neg Q \vee \neg S, \neg R \vee S, P \vee Q \vee R, \neg P\}$ in rule notation becomes: $\{Q \wedge S \rightarrow \text{false}, R \rightarrow S, \text{true} \rightarrow P \vee Q \vee R, P \rightarrow \text{false}\}$. The only possible hyper tableau for f_N and S_2 is as follows:



Note that the left branch is closed. A hyper tableau for S_2 must begin with the only restart clause. Closure of the leftmost branch and extension of the rightmost one are both mandatory. No other rule can be applied after this.

In general, hyper tableaux are not complete (for example, when *all* literals are selected no initial extension step can be made), but completeness is regained for consistent selection functions.¹¹

4.22. THEOREM. *If the finite ground clause set S is unsatisfiable, then for any consistent selection function f there is a hyper tableau proof with selection function f for S .*

PROOF. The proof is very similar to the proof of Theorem 4.18. Consider an open branch B in a hyper tableau for S , in which all possible extension steps were made, and let \tilde{B}_f be the extension of B , see (4.3). The set $\tilde{B}_f - B$ is consistent, because f is consistent, so when \tilde{B}_f is inconsistent there is a literal $L \in \{f(C) \mid C \in S, \mathbf{I}_B \not\models C\}$ such that \bar{L} occurs in B . Obtain B' by removing all such literals from \tilde{B}_f . Now B' is consistent and we claim $\mathbf{I}_{B'} \models S$. By construction $B' \supseteq B$, so it suffices to prove $\mathbf{I}_{B'} \models C$ for clauses C not used on B (this implies $|f(C)| > 0$).

\tilde{B}_f contains the literals from $f(C)$; if all of these were removed when computing B' then $\overline{f(C)} \subseteq B$ and a hyper extension step would be possible with C on B

¹¹In fact, for consistent selection functions that select at most one literal per clause conditions (4.4) and (4.5) become identical.

contradicting the assumption that all possible extension steps were made. So at least one literal in $f(C)$ is still present in B' , thus $\mathbf{I}_{B'} \models C$. \square

The set B' that induces the model $\mathbf{I}_{B'}$ in the previous proof can be defined more directly as $B' = B \cup \{L \in f(C) \mid C \in S, \bar{L} \notin B\}$.

4.6.1. Positive and Semantic Hyper Tableaux

Many hyper tableau calculi [Manthey and Bry 1988, Fujita and Hasegawa 1991, Brown 1978, Baumgartner et al. 1996, Bry and Yahya 1996, Shults 1997, Baumgartner 1998] focus on the particular selection function f_N which selects exactly the negative literals in a clause. The ensuing calculi are called *positive hyper tableaux*, because negative literals occur only as leaves of branches closed by an extension step in a hyper tableau with f_N . Therefore, closure by reduction steps cannot occur. If Σ contains the atoms of S , then computing B' simplifies to $B \cup \{\neg P \mid P \in (\mathcal{A}_\Sigma - B)\}$. From the two open branches in the tableau of Example 4.21 one reads off the models $\{Q, \neg S, \neg R, \neg P\}$ and $\{R, S, \neg Q, \neg P\}$.

Selection function f_N is an example of a *complete selection function*¹², which selects at least one of $P, \neg P$ for each ground atom in the signature of ground clause set S .¹³ In the presence of complete selection functions reduction steps are not required, because they would occur with two complementary, not selected literals whose existence is exactly what has been ruled out;¹⁴ this proves:

4.23. COROLLARY. *If the finite ground clause set S is unsatisfiable, then for any consistent and complete selection function f there is a hyper tableau proof with selection function f for S that does not use the reduction rule.*

4.24. EXAMPLE. Completeness of the selection function is a necessary condition in the corollary: consider the clause set $S = \{\text{true} \rightarrow P, \text{true} \rightarrow \neg P\}$ in which nothing is selected (that is, its corresponding selection function is consistent). Both clauses are restart clauses, so P and $\neg P$ are present on any branch via extension, however, closure is only possible with a reduction step.

4.6.2. First-Order Issues

For sake of clarity, the first-order version of the *hyper tableau extension* rule is stated explicitly:

¹²This notion of completeness is unrelated to completeness of calculi.

¹³Consistent and complete selection functions on S can be considered as interpretations of S ($\mathbf{I}_f \models L$ iff $L \in f(C)$). Hyper tableaux based on such f can be seen as a tableau counterpart to *semantic resolution* [Slagle 1967] and would be best called *semantic tableaux* if the latter phrase were not used sometimes for the whole tableau framework. Perhaps one should call them *semantic semantic tableaux*?

¹⁴Alternatively, one argues that consistent and complete selection functions f induce a suitable literal renaming of S on which f_N can be used to the same effect as f .

(ii''') Let T be a hyper tableau for S and consistent selection function f , B a branch of T , $C = \overline{L_1} \wedge \cdots \wedge \overline{L_m} \rightarrow L_{m+1} \vee \cdots \vee L_n$ a new instance of a clause in S . If there is an MGU σ such that all literals of the premise of $C\sigma$ occur in $B\sigma$ (see also (4.7)), $I_{\exists B\sigma} \not\models \forall C\sigma$, and the tree T' is constructed by extending B with r new subtrees, where the nodes of the new subtrees are the literals of C , then $T'\sigma$ is a hyper tableau for S and f , in which all new branches ending with $L_i\sigma$ ($1 \leq i \leq m$) are marked as closed.

As before, selection functions must be liftable, that is, stable with respect to substitutions to ensure completeness. As can be seen from (ii'''), in addition to the regularity and tautology check (that is, $I_{\exists B\sigma} \not\models \forall C\sigma$), one must compute a set $B_0 \subseteq B$ as well as a simultaneous MGU of $\{\{L, L'\} \mid L \in B_0, L' \in \overline{f(C)}\}$ to perform a hyper extension step on branch B with clause C . This is easily seen to be an instance of the first-order clause subsumption problem, which is NP-complete [Garey and Johnson 1979]. This complexity is implicitly present in proof search without hyper steps as well, although on a different level. It does not indicate inferiority of hyper tableaux as a calculus for proof search.

A more interesting question takes up the discussion of Section 3.3: how deals one in the context of hyper tableaux with the difficulties to define combined fair clause and substitution selection in destructive calculi? For positive hyper tableaux, several strategies are found in the literature:

The first option is to fix a computation rule and accept incompleteness. Typically, one minimizes $n - m$ in (ii''') and/or the size of terms in MGUs. For specific problem domains, where specific heuristics could be developed, success is reported in [Brown 1978, Shults 1997]. Another form of incompleteness (in syntactical expressivity of the logic) ensues from restriction to range-restricted sets of clauses:

4.25. DEFINITION. A first-order clause C is *range-restricted* with respect to a selection function f when all variables occurring in the conclusion of C (that is, in not selected literals) occur also in its premise (that is, in selected literals).

Observe that range-restricted clauses with no selected literals (premise is $\{true\}$) are ground. A trivial induction shows that when all clauses in S are range-restricted, then a hyper tableau for S is ground. As an immediate consequence, in this case hyper tableaux are not destructive, so fair selection of clauses used in extension steps renders hyper tableaux a strongly complete calculus, of which efficient implementations were realized with respect to f_N [Manthey and Bry 1988, Fujita and Hasegawa 1991, Hasegawa, Koshimura and Fujita 1992, Bry and Yahya 1996, Hasegawa, Fujita and Koshimura 1997] (by Corollary 4.23 reduction steps are not necessary). These calculi are further discussed in Section 4.6.3 below.

EP-tableaux are a variant of positive hyper tableaux for non-clausal first-order logic without function symbols that can detect finite satisfiability [Bry and Torge 1998]. The key ingredients are (i) a non-clausal version of range-restrictedness called *positive formulas with restricted quantifications* (PRQ formulas)—in particular, type γ formulas are of the form $(\forall \vec{x})(\phi(\vec{x}) \rightarrow \psi)$ and are only used on a branch

B with $I_B \models \phi\sigma$ for some σ ; (ii) only formulas ψ with $I_B \not\models \psi$ are added to a branch B , (iii) a modified δ -rule that explicitly enumerates the current model domain:

$$\frac{\delta(x)}{\delta_1(c_1) \mid \cdots \mid \delta_1(c_m) \mid \delta_1(sko_\delta)}$$

c_1, \dots, c_m are all
constants on the branch.

Because of (i), EP-tableaux have no free variables, so they are not destructive. In the function-free case, finite term domain models have a finite representation on branches; (iii) ensures that enough information for explicit model construction is present. Together with (ii) and a fair computation rule, this is sufficient to detect all finite term domain models on finite EP-tableau branches.

Back to general clause sets, call a variable violating range-restrictedness of a clause *and* occurring in more than one literal of the conclusion of this clause *critical*. Obviously, non-critical variables in a tableau are *universal* in the sense of Section 3.2.5 and need, therefore, not be instantiated. Baumgartner et al. [1996] enumerate ground instances of clauses restricted to critical variables to obtain a strongly complete calculus which is better than enumerating all ground instances as in Smullyan's [1995] tableaux.

Baumgartner [1998] improves on this: like in the range-restricted case, a tableau is treated as if it were ground: substitutions are only applied to new instances of input clauses (in other words, not unification, but merely matching is employed). If an instance $L\sigma$ (with critical variables) of a literal occurrence L on a tableau branch is required to perform an extension or reduction step, then σ is applied to an instance of the clause containing L and the result is added to the input clause set. The latter possibility regains completeness.¹⁵ Needless to say, lifting is not trivial in such a calculus.

This version of first-order hyper tableaux is not destructive, as only input clauses are instantiated. The destructive part of the substitution of the closure rule is recorded "outside" of the tableau as additional instances of input clauses. They can be arranged in a fair manner easily. This can also be seen as a kind of constraint on substitutions to guide proof search.

In principle, a strongly complete, *destructive* calculus could be obtained if one compiled the information contained in these "outside" clauses (and their fair selection) into a clause selection rule. Consequences for such a calculus would be: (I) clause selection is not defined branch-local, because "outside" clauses touch on several branches; (II) one needs to identify clause instances that are identical up to variable renaming to ensure finiteness; (III) a suitable ordering on literals must be used to enumerate instances of literals for closure in a fair manner. All three ingredients are actually present in recent suggestions for strongly complete, destructive proof procedures in [Beckert 1998, Beckert 2000] and [Baumgartner, Eisinger

¹⁵This idea can be applied to any tableau calculus, not only to hyper tableaux. They are particularly suitable, though, because less clause instances are generated.

and Furbach 1999, Baumgartner, Eisinger and Furbach 2000], the first of which is discussed in Section 4.7.

4.6.3. Model Generation

Positive hyper tableaux for range-restricted clause sets are better known as *model generation* and were suggested by Manthey and Bry [1988]. Traditionally, they are described in a somewhat different manner (adapted from [Fujita and Hasegawa 1991]):

4.26. DEFINITION. \mathcal{M} is an inductively defined set of interpretations called **model candidates** each of which is represented by a consistent set of ground literals.

Init: Set \mathcal{M} to $\{\emptyset\}$

Model Extension: $I \in \mathcal{M}$ is a model candidate, $D \rightarrow E$ a new instance of a clause in S . If there is a substitution σ such that $D\sigma \subseteq I$ and $I \not\models E\sigma$, then set \mathcal{M} to

$$(\mathcal{M} - \{I\}) \cup \{\{L\} \cup I \mid L \in E\sigma\} .$$

Model Rejection: $I \in \mathcal{M}$ is a model candidate, $D \rightarrow \text{false}$ a new instance of a clause in S . If there is a substitution σ such that $D\sigma \subseteq I$, then delete I from \mathcal{M} .

Model candidates correspond to open hyper tableaux branches, while model extension and model rejection are special cases of the positive hyper tableau extension rule. Model candidates that can neither be extended nor rejected correspond to open hyper tableau branches with no applicable rule and, therefore, induce a model of the input clause set. A closed tableau corresponds to the empty set of model candidates.

In the light of Corollary 4.23, model generation works unaltered for range-restricted rule sets with consistent and complete selection functions, if selection functions are suitably defined on the first-order level:

4.27. DEFINITION. A selection function f is *consistent* on a first-order clause set S if it is consistent on all its ground instances. It is *complete* on S if for each ground instance P of an atom occurring in S : f selects a literal Q or $\neg Q$ such that P is an instance of Q .

A further generalization of model generation was obtained by Shirai and Hasegawa [1995] and called *constraint model generation*. It can be described in the present framework as hyper tableaux with arbitrary selection function and range-restricted input. Theorem 4.22 (plus a trivial lifting step) grants completeness for fair input clause selection and consistent selection functions, when a reduction rule is present. But the latter can be expressed *within* the calculus by adding a clause of the form

$$P(x_1, \dots, x_n) \wedge \neg P(x_1, \dots, x_n) \rightarrow \text{false} \quad (4.8)$$

for each n -ary predicate symbol $P \in P_\Sigma$. The rules (4.8) were called *integrity constraints* by Shirai and Hasegawa.

Whether constraint model generation is complete for a given problem S was unclear in [Shirai and Hasegawa 1995]. From the tableau perspective, Theorem 4.22 ensures completeness for each problem S with a consistent set of rule premises (with the exception of the problem-independent rules (4.8)). The quasi-group problems discussed in [Shirai and Hasegawa 1995] are a practically relevant case.

If necessary, any clause set can be made consistent with respect to any given selection function f : assume there is a rule $R = L \wedge C \rightarrow D$ such that L is unifiable with \bar{L}' occurring in the premise of another rule. Replace R with $R' = d(\vec{x}) \wedge C \rightarrow D \vee \bar{L}'$, where \vec{x} are the variables occurring in L but not in C , and d is a *domain predicate* that enumerates the ground terms of the problem signature [Manthey and Bry 1988]. This preserves satisfiability and range-restrictedness and eliminates the inconsistency at L .

4.7. A Destructive and Strongly Complete First-Order Calculus

Recall from the discussion in Section 3.3 that destructive first-order tableau calculi cannot easily be turned into a strongly complete proof procedure so that one usually retracts to DFID search, although backtracking is not necessary, in principle, within proof confluent calculi.

Recall further that it is the closure rule (Definition 4.2(iii)) that renders tableaux with unification destructive, but the MGUs computed in it are needed for guidance of proof search.

Baumgartner's [1998] idea, briefly discussed in the previous section, is to record MGUs outside of the tableau in the form of a dynamically growing input clause set. Beckert [1998] has a different approach not modifying the input clause set, but the tableaux themselves.¹⁶

Whenever a substitution σ must be applied to close a branch in a tableau T , the smallest subtableau T' of T affected by σ (that is, containing variables x with $\sigma(x) \neq x$) is *reconstructed*. This can be done trivially by copying T' below each open branch of $T'\sigma$ as depicted in Figure 12, but obviously less redundant strategies are conceivable.

The effect is that substitution is syntactically still destructive, but from a semantical point of view absolutely no harm is done.

Select clause and *select pair* must still be fair, of course, but this is much easier to achieve once destructivity is essentially eliminated. Unfair selection can result in two phenomena which have to be both avoided: (i) generating arbitrarily large terms of one kind (such as $s^n(x)$ for all n , when also, say, 0 is present); (ii) avoid loops of tableaux that subsume each other.

To avoid unfair generation of terms one restricts the order in which they can be introduced. A *well-order* $<$ on literals is any partial, well-founded order, such that there are not infinitely many incomparable elements (up to variable renaming).

¹⁶Beckert's framework is designed for quite general tableaux calculi including non-classical and non-clausal logics. I present it in simplified form for clause logic, because the technicalities of the general case are beyond the scope of this article.

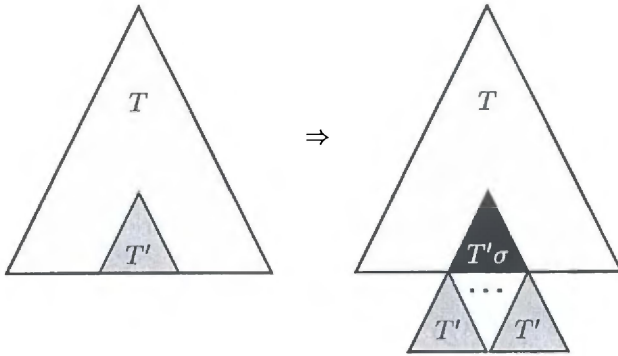


Figure 12: Reconstructing a subtableau “destroyed” by substitution.

4.28. **EXAMPLE.** Let $|L|$ denote the number of symbols in L , where variables are counted twice. Then define a well-order $<$ by $L < L'$ iff $|L| < |L'|$ and $false < L$ for all L . This implies $P(a) < P(x) < \neg P(y)$.

Now the effect of each rule, when applied to a tableau, can be measured in terms of $<$:

4.29. **DEFINITION.** With each tableau construction rule R applied to tableau T one associates a set of literals $R(T)$, depending on the rule type:

Extension: the set of literals in the clause instance used for extension;

Closure: $\{false\} \cup \{L\sigma \mid L \text{ occurs in } T, L\sigma \neq L\}$, when closure is by MGU σ .

It remains to assure that each tableau rule application derives “new” information so that progress towards a proof is not infinitely delayed. In tableau calculi progress can be measured in terms of branch closure. If a tableau can be closed, and it turns out that its predecessor could have been closed as well, then this tableau is not needed. The following definition formally captures this:

4.30. **DEFINITION.** A tableau T *subsumes* a tableau T' if each branch of T subsumes at least one branch of T' .

A branch B of T *subsumes* a branch B' of T' if for all sets Φ' of literals on B' with at most two elements there is a set Φ of literals on B such that

1. $\Phi\pi = \Phi'$ (where π is a variable renaming)
2. for each literal L in T that has variables in common with Φ there is an L' in T' such that $L\pi = L'$ up to renaming of free variables not occurring in Φ' .

The reason for $|\Phi| \leq 2$ is that a single rule application involves either one (extension) or two (reduction) branch literals. The second, rather technical, condition is required to guarantee that a deduction that is possible in T' , can be mimicked in T , and has the same effect on instantiations and order. Putting things together, one obtains:

4.31. THEOREM ([Beckert 1998]). *If S is an unsatisfiable first-order clause set and $<$ a well-order on literals, then any sequence of first-order clause tableaux for S results in a tableau proof for S after finitely many steps provided that (A) a reconstructing version of the closure rule is used and (B) a rule R is only applied to a tableau T when*

1. *the $<$ -maximal elements of $R(T)$ are $<$ -minimal in $\bigcup_{R' \text{ applicable on } T} \max R'(T)$*
2. *T does not subsume the result of applying R to T .*

4.32. EXAMPLE. Consider $S = \{P(a), Q(a), Q(b), \neg P(x) \vee \neg Q(x)\}$ and the well-order defined in the previous example. Starting with the initial tableau, the ground clauses must be applied first, because closure is not possible and ground literals are minimal. No ground clause can be applied twice, because the resulting tableau would be subsumed. The top left hand tableau in Figure 13 is obtained.

Assume *select branch* is implemented right-first. Now a second application of the non-ground clause competes with closure as indicated (the substitution that does not immediately lead to a proof is chosen deliberately). The literal set associated with the closure contains only ground literals and so is preferred over extension. Substitution $\{x \mapsto b\}$ affects the tableau below $Q(b)$ which is, therefore, replicated. The reconstructed tableau is on the right on the top row.

In the rightmost open branch, again, closure is preferred over extension. But if the same substitution as before is used, then the resulting tableau (right hand in bottom row) is subsumed by the top right tableau: the unchanged branch is trivially subsumed; *both* gray branches subsume the same gray branch in the subsumed tableau.

This leaves closure with $\{x \mapsto a\}$ as the only legal rule application which results in a tableau proof immediately.

4.8. Tableaux with Cuts and Lemmas

So far we discussed restrictions of tableau calculi aimed at diminishing the search space. Some problems, however, only have extremely long tableau proofs. Already on the ground level there exist classes of formulas S_n such that the size of their smallest tableau proof is exponential in the size of S_n whereas short resolution proofs exist [Cook and Reckhow 1979]. The reason is that resolution incorporates an atomic cut rule or (and this is just another name) lemma generation.

4.33. EXAMPLE. Let $\{P_1, \dots, P_n\}$ be different ground atoms. Consider the clause set

$$S_n = \{L_1 \vee \dots \vee L_n \mid L_i \in \{P_i, \neg P_i\}, i \in \{1, \dots, n\}\} \quad (4.9)$$

Obviously, S_n is unsatisfiable. D'Agostino [1992] proved that the smallest closed clause tableau for S_n has at least $n!$ inner nodes, whereas S_n contains merely $n2^n$ literals. Even simple truth table checking has linear cost in the size of S_n .

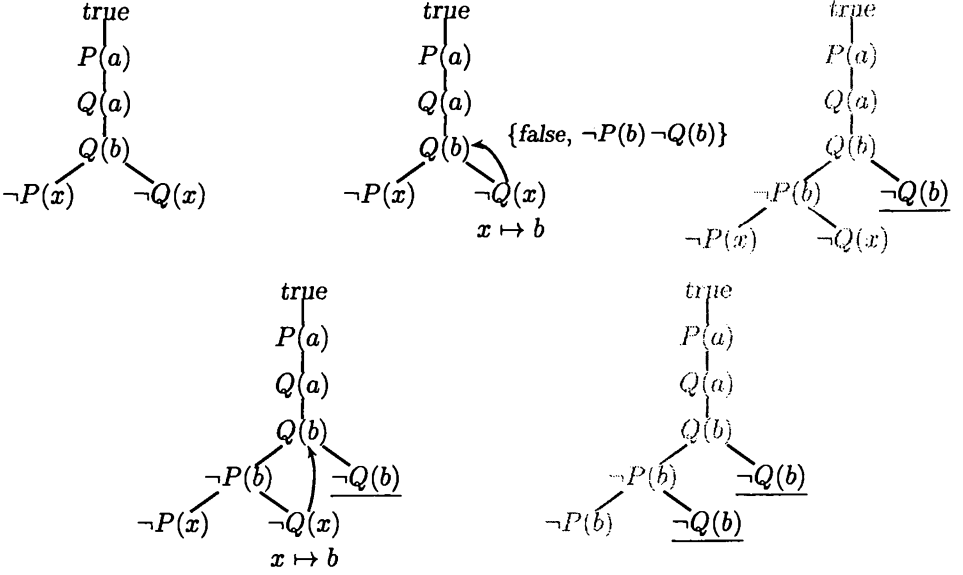
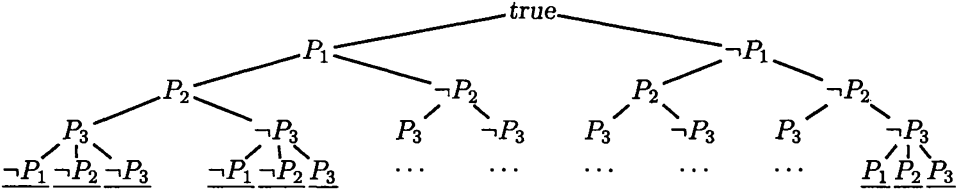


Figure 13: Illustration of Example 4.32.

Now consider the clause set $T_n = \{P_i \vee \neg P_i \mid i \in \{1, \dots, n\}\}$. $S_n \cup T_n$ has a short proof (displayed for $n = 3$ in Figure 14). The point is that the tautologies in T_n can be used to enumerate all interpretations over $\{P_1, \dots, P_n\}$. Then each clause in S_n is contradicted by exactly one interpretation. The resulting tableau has $n2^n + 2^{n+1} - 1 \in \mathcal{O}(\|S_n \cup T_n\|)$ nodes.

Figure 14: Clause tableau proof for $S_3 \cup T_3$.

The effect of the clauses T_n can also be achieved by adding a new tableau extension rule

$$\frac{}{P \quad | \quad \neg P} \quad (4.10)$$

called *atomic cut* rule. It is closely related to the well-known cut rule found in sequent calculi [Gentzen 1935]. This is what we look at next.

Clausal Sequent Calculus	Clausal Tableau Calculus
Axiom rule	Branch closure rule
\vee -left rule	Tableau extension
Atoms/unit clauses left of ' \Rightarrow '	Positive branch literals
Atoms right of ' \Rightarrow '	Negative branch literals
Sequent proof trees interpreted as logical conjunction of their leaf sequents	Tableaux interpreted as logical disjunction of their branches
Sequents interpreted as logical disjunction of their elements	Branches interpreted as logical conjunction of their literals
Validity proof	Unsatisfiability proof

Table 3: Duality between sequent and tableau calculi.

4.8.1. Tableaux and Sequent Calculi

Sequent calculi [Gentzen 1935] are direct ancestors of tableaux; see [Avron 1993, Smullyan 1995] for full accounts of their relationship, which I sketch here for the clausal case.

A *propositional clausal sequent* is an expression of the form $\Gamma \Rightarrow \Delta$, where Γ is a tuple of clauses and Δ is a tuple of atoms. $\Gamma \Rightarrow \Delta$ is *valid* iff the formula $\bigwedge_{G \in \Gamma} G \rightarrow \bigvee_{D \in \Delta} D$ is valid. Hence, a clause set S is unsatisfiable if the sequent $S \Rightarrow$ is valid. The propositional clausal sequent calculus consists of only three rule schemata:

$$\begin{array}{c}
 \frac{\Gamma, L_1, \Gamma' \Rightarrow \Delta \quad \cdots \quad \Gamma, L_n, \Gamma' \Rightarrow \Delta}{\Gamma, L_1 \vee \cdots \vee L_n, \Gamma' \Rightarrow \Delta} \vee\text{-left} \\
 \frac{\Gamma, \Gamma' \Rightarrow P, \Delta}{\Gamma, \neg P, \Gamma' \Rightarrow \Delta} \neg\text{-left} \quad \frac{\times}{\Gamma, P, \Gamma' \Rightarrow \Delta, P, \Delta'} \text{axiom}
 \end{array} \tag{4.11}$$

Sequent proof trees have sequents as their nodes. A sequent proof tree for a sequent $\Gamma \Rightarrow \Delta$ has this sequent as its root and is extended by applying suitable instances of rule schemata (4.11) to leaves.

It is straightforward to show that a sequent is valid iff it has a proof tree in which all leaves are marked with \times . Moreover, there is a duality between clause tableaux and clausal sequent calculi, summarized in Table 3.¹⁷

Literal occurrences may be shared among several tableau branches, but are duplicated in sequent proof trees. In the light of this and the duality between sequent and tableau proofs, rule (4.10) corresponds exactly to the usual cut rule of sequent calculi, if the *cut formula* ϕ is restricted to being an atom:

¹⁷This duality extends to the non-clausal and first-order case, see [Smullyan 1995].

$$\frac{\Gamma, \Gamma' \Rightarrow \Delta, \phi, \Delta' \quad \Gamma, \phi, \Gamma' \Rightarrow \Delta, \Delta'}{\Gamma, \Gamma' \Rightarrow \Delta, \Delta'} \quad (4.12)$$

4.8.2. Tableaux with Lemmas

We saw that the atomic cut rule (4.10) can lead to exponentially shorter tableau proofs, however, its application is completely unrestricted—one can use it anytime during tableau construction. This may well cause longer proofs than shorter ones. It is not obvious when to apply cut advantageously. A first restriction is obtained by permitting its use only if the extension rule application immediately following it during tableau construction is a connected extension step (see Figure 15).

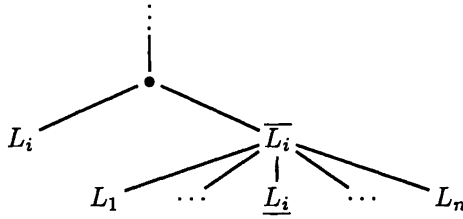


Figure 15: Cut rule application followed by connected extension step.

With this restriction in place, not more branches are generated than if one had performed only the extension step without the preceding cut. The cut has the effect that in all $n - 1$ branches that contain a literal $L_j \neq L_i$ in addition the literal $\overline{L_i}$ is present. If one applies $n - 1$ atomic cuts to $n - 1$ different literals from $\{L_1, \dots, L_n\}$ before an extension step and writes the resulting proof tree as a “macro rule” (not displaying closed branches) yields the *extension rule with local lemmas* displayed in Figure 16.

$L_{\pi(1)}$	$\overline{L_{\pi(1)}}$	$\overline{L_{\pi(1)}}$	\dots	$\overline{L_{\pi(1)}}$	$L_1 \vee \dots \vee L_n \in S$ π permutation of $\{1, \dots, n\}$
	$L_{\pi(2)}$	$\overline{L_{\pi(2)}}$	\dots	$\overline{L_{\pi(2)}}$	
		$L_{\pi(3)}$	\dots	$\overline{L_{\pi(3)}}$	
				\vdots	
			\ddots	$\overline{L_{\pi(n-1)}}$	
				$L_{\pi(n)}$	

Figure 16: Extension rule with local lemmas.

The complemented literals are called *local lemmas*: for example, $\overline{L_{\pi(i)}}$ is obtained as a lemma on its sibling branches after the branch B is closed with the help of

$L_{\pi(i)}$. (Recall that closed branches are unsatisfiable, hence, a branch that can be closed and where $L_{\pi(i)}$ occurs, proves $B \models \overline{L_{\pi(i)}}$.) The lemma is *local* to certain branches as opposed to being global (on the whole tableau).

4.8.3. Tableaux with Folding Up

Depending on the sequence of branch closures, there are $n!$ different permutations of the local lemma rule applied to a n -literal clause. Not all of these are equally useful. To remedy this situation, a version of local lemma generation called *folding up rule* has been suggested [Letz 1993]. It can be seen as “lazy lemma generation”.

4.34. EXAMPLE. Consider the clause set $S = \{P \vee T, P \vee \neg T, \neg P \vee Q \vee S, \neg Q \vee R, \neg R \vee \neg P, \neg S \vee R\}$ and the partial tableau proof for S displayed in Figure 17. Assuming that *select branch* is left-first, after closure of branch $B = \{P, Q, R\}$ one knows that $S \cup \{P, Q\} \models \neg R$. This lemma is useless, though, because the branch $\{P, Q, \neg Q\}$ to the left of B is closed already, and B has no right sibling. Inspection of the partial proof shows, however, that even $S \cup \{P\} \models \neg R$ holds, because Q was not involved in the closure of B .

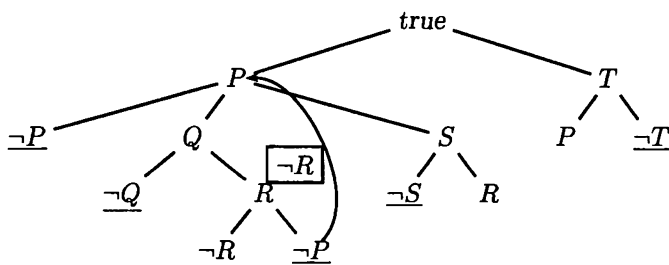


Figure 17: Illustration of Example 4.34.

In clause tableaux with *folding up rule* each literal L that is a local lemma in the sense of the local lemma rule in Figure 16 may be moved up towards the root as follows: we say that a literal L' on the path between L and the root was *used to prove* L if L' is involved in the closure of a branch through L' ; now the folding up rule permits to move L immediately above the lowest literal used to proof L [Letz 1993, Letz et al. 1994] and it can be used to close any branch below its new position.

With the folding up rule, in the previous example, lemma $\neg R$ may be moved up above P and lemma $\neg P$ from the right subtree is moved up above $true$ (the latter is also possible with a suitable variant of the local lemma rule). In Figure 18 new lemma positions are boxed, upward moves are indicated by dashed arrows, while additional closures made possible by moved lemmas are indicated by solid arrows.

One can show that the folding up rule does not change the *nondeterministic power* of tableau with local lemmas, that is, the length of shortest proofs does not

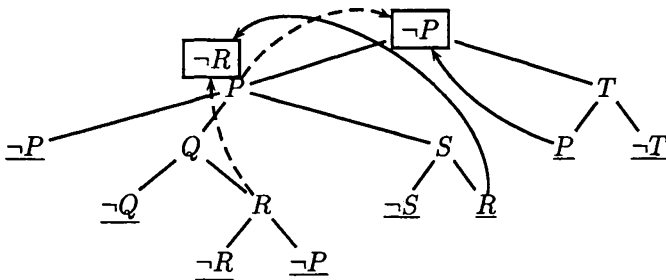


Figure 18: Tableau with folding up rule.

change [Letz 1993]. In a concrete tableau proof search the shortest proof is generally not found, therefore, folding up can speed up proof search considerably: an ill choice of *select clause* might be remedied.

4.8.4. Tableaux with Factoring

Tableaux with factoring (also called *tableaux with merging*) are related to the subsumption rule in resolution theorem proving [Robinson 1965]. Assume branch B is not yet closed, but there is a “more specific” branch B' closed already. Intuitively, this justifies to close B as well (as long as one does not go around in circles). Formally, a branch B' is *more specific* than a branch B iff there is a substitution σ such that $B' \subseteq B\sigma$. Instead of trying to close B , one may simply refer to B' then, apply σ to B and consider it as closed provided that these references are acyclic. Soundness of tableaux with factoring is straightforward to prove directly; alternatively, it is sufficient to observe that tableaux with factoring can be simulated by tableaux with local lemmas: assume B was closed by referring to the more specific branch B' and that L is the top-most literal on B' not on B . Now replace the extension rule that produced L with a local lemma version putting \bar{L} on B . As B' is more specific than B , there must be $L' \in B$ and substitution σ such that $L'\sigma = L$, so B can be closed with σ .

An improvement of tableaux with factoring called *tableaux with regressive merging* was introduced in [Wallace and Wrightson 1995] and is essentially the same as tableaux with the folding up rule; details can be found in [Wallace 1994].

4.8.5. Problems of Strengthening Tableaux

Strengthening of tableau procedures at first seems a sure win, because length of proofs can be drastically reduced. On the propositional level this holds without reservation and it can safely be claimed that any competitive propositional proof procedure embodies some variant of cut. On the first-order level, however, additional literals introduced as cuts or lemmas create additional possibilities for branch closure. The negative effects from this increase of the search space can easily outweigh the possibility of finding shorter proofs.

4.35. EXAMPLE. $S = \{\neg P(x) \vee Q(x) \vee R(x), P(a), \neg Q(a), P(b), \neg Q(b), \neg R(b)\}$. In the partial connection tableau in Figure 19 the left branch was closed first by extension with $P(a)$ (only $P(b)$ would lead to success). This forces extension with $\neg Q(a)$ in the middle branch and generation of some lemmas (framed literals). The lemmas allow to extend the right branch with another instance of the first clause which would have been impossible without them. Detection of the wrong first extension is thus possibly delayed a long time.

In the example, a regularity check would help ($R(a)$ becomes irregular on the rightmost branch) and also restriction of lemma usage to reduction steps. Although this helps somewhat, more complex examples create the same problems as before.

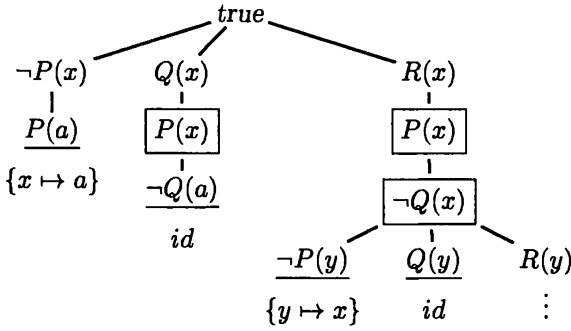


Figure 19: Search space increase caused by local lemmas.

On the other hand, local lemmas are not strong enough on the first-order level. In the clause tableau for $S = \{Q(c) \vee Q(d), \neg Q(x) \vee P, \neg P\}$ in Figure 20, for example, the lemma $\neg Q(c)$ can be folded up to the *true* node, but it cannot be used to close the open branch on the right, because a different instance than c is required. Closer inspection of S , however, shows that it is in fact justified to derive the stronger lemma $(\forall x)\neg Q(x)$. A sufficient condition for lemma generalization is reached when the proof of (that is, the tableau below) the node giving rise to the lemma does not instantiate any variables that occur elsewhere in the tableau. It remains to be seen whether such an optimization can be efficiently implemented and whether it does not blow up the search space beyond any usefulness.

5. Tableaux as a Framework

In this section it is argued that many well-known calculi for automated deduction can be uniformly presented within a tableau framework and that this deepens the understanding of these calculi. In Section 4.6 I put model generation into a tableau perspective. The relationship between tableaux and the connection method [Bibel 1982b] is discussed in Section 3.5 above for the general, non-clausal case. Therefore, these calculi are not discussed again in the following.

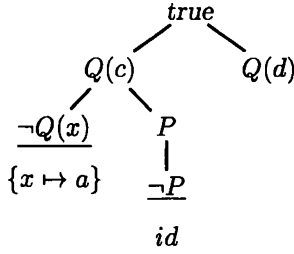


Figure 20: Insufficient power of local lemmas.

5.1. Model Elimination

Model elimination¹⁸ was suggested by Loveland [1968b, 1969] as a space efficient calculus with small local search space for first-order clausal deduction. The following, slightly simplified, definition is taken from his [1969] article.

5.1. DEFINITION. A *chain* is a finite sequence of literals $\langle L_1, \dots, L_n \rangle$ each of which has either *type A* or *type B*. (Type A literals will be framed for easy distinction.)

Define $\langle L_1, \dots, L_n \rangle \sigma = \langle L_1 \sigma, \dots, L_n \sigma \rangle$, then construction rules for *admissible chains* for a given set of first-order clauses S are as follows:

Init: If $L_1 \vee \dots \vee L_n$ is a new instance of a clause in S , then the chain $\langle L_1, \dots, L_n \rangle$ consisting of type B literals is admissible.

Extend: If $\langle L_1, \dots, L_n \rangle$ is an admissible chain, L_n of type B, $C = M_1 \vee \dots \vee M_m$ a new instance of a clause in S , and there is an MGU σ such that $\overline{L_n} \sigma \in C \sigma$, then $\langle L_1, \dots, L_{n-1}, \boxed{L_n}, M_1, \dots, M_{j-1}, M_{j+1}, \dots, M_m \rangle \sigma$ is admissible, where the M_i are of type B.

Reduce: If $\langle L_1, \dots, L_n \rangle$ is an admissible chain, $\boxed{L_i}$ of type A, L_j, \dots, L_n of type B ($i < j$), and there is an MGU σ of $\{\overline{L_i}, L_j\}$ then $\langle L_1, \dots, L_{j-1}, L_{j+1}, \dots, L_n \rangle \sigma$ is admissible.

Contract: If $\langle L_1, \dots, \boxed{L_n} \rangle$ is an admissible chain and $\boxed{L_j}, \dots, \boxed{L_n}$ are of type A, then $\langle L_1, \dots, L_{j-1} \rangle$ is admissible.

Type A literals can be memorized as *ancestor literals*.

5.2. DEFINITION. A *model elimination* proof of a clause set S is a sequence of admissible chains for S ending with the empty chain $\langle \rangle$.

5.3. THEOREM ([Loveland 1969]). *A clause set S is unsatisfiable iff it has a model elimination proof.*

¹⁸A detailed comparison between model elimination and tableaux on a different basis than here is contained in [Baumgartner and Furbach 1993].

5.4. EXAMPLE. Consider $S = \{P, R \vee \neg P \vee Q, S \vee \neg Q, \neg Q \vee \neg S, \neg Q \vee \neg R \neg R \vee Q\}$. A model elimination proof of S is as follows:

0. P Init
1. \boxed{P} R Q Extend
2. \boxed{P} R \boxed{Q} S Extend
3. \boxed{P} R \boxed{Q} \boxed{S} $\neg Q$ Extend
4. \boxed{P} R \boxed{Q} \boxed{S} Reduce
5. \boxed{P} R Contract
6. \boxed{P} \boxed{R} Q Extend
7. \boxed{P} \boxed{R} \boxed{Q} $\neg R$ Extend
8. \boxed{P} \boxed{R} \boxed{Q} Reduce
9. Contract

For sake of readability delimiters of chains are not displayed, lines are numbered starting with 0. The initial step can be with any clause from S : take the first. Now P is the single type B literal. Neither reduction nor contraction is allowed. The only possibility to extend is with the second clause, because a clause with an occurrence of $\neg P$ is required. The type B literal P turns into a type A literal and $\neg P$ is deleted.

Again, only extension is possible and a clause with $\neg Q$ in it is required of which the first is chosen. Then extend with $\neg S \vee \neg Q$, the result is line 3.

Finally, a reduce step is possible with \boxed{Q} ; type B literal $\neg Q$ is deleted. The rightmost block of type A literals is removed with the contract rule.

Two extension steps with $\neg R \vee Q$ and $\neg Q \vee \neg R$ are followed by reduction with \boxed{R} , whereby $\neg R$ is deleted. The empty chain is obtained by contraction of all remaining (type A) literals.

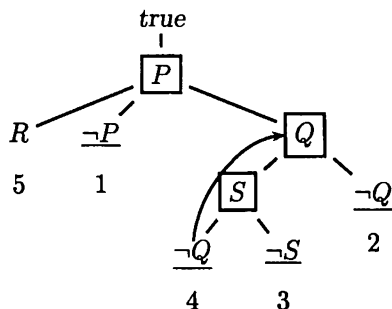


Figure 21: Partial regular connection tableau corresponding to the model elimination proof in Example 5.4.

Model elimination is closely linked to the connection tableau calculus. This is demonstrated by the tableau in Figure 21 which simulates lines 0 to 5 in the model

elimination proof above. The number below each branch indicates which line in the model elimination proof corresponds to its closure. Type *A* literals correspond to inner tableau nodes. Type *B* literals occurring between two type *A* literals in chains correspond to leaves of branches yet to be closed. Obviously, the initialization, the extension, and the reduction rule correspond to tableau rules with the same name. The contract rule changes the focus from closed branches without open siblings to the next open branch to the left. Hence, a chain is simply a linear format to represent a partial connection tableau. This is possible, because *select branch* in model elimination chooses always the rightmost branch for extension.

In the paper [Loveland 1969, p. 351] chains are restricted in a way that corresponds to regularity in tableaux. Two different completion modes of chains for a given clause set are specified which vary the necessary number of extensions to produce a chain. A lemma generation mechanism that corresponds to local lemmas in tableaux is described as well.

Like matrices [Andrews 1981, Bibel 1981], model elimination chains are a much more implementation-oriented format than tableau trees. This makes it harder to see further optimizations (such as more liberal branch selection rules or folding up) and to prove completeness. The latter follows directly from completeness of regular connection tableaux, of course.

Model elimination can be interpreted as a systematic way to exclude certain interpretations as possible models of a clause set [Loveland 1969, p. 362], but it should be stressed that those type *A* literals in a non-empty chain to which no rule can be applied in general do *not* constitute a partial model (or have a saturation, in the terminology of Section 4.5), because model elimination, just like connection tableaux, is not proof confluent. To summarize, model elimination (without the lemma generation mechanism) can be conceived as a notational variant of regular connection tableaux (although it preceded the latter, of course).

5.2. Linear Resolution

Linear resolution [Loveland 1968a, Luckham 1968] and its refinements do not fit directly into the framework of standard resolution [Robinson 1965], because their definition, like that of tableau calculi with connection condition, relies on the form of the derivation. Moreover, in first-order SL-resolution [Kowalski and Kuehner 1971, Reiter 1971] (discussed on page 69 in Chapter 2 of this Handbook) variables are treated rigidly like in tableaux with unification. Indeed, as Loveland [1972] observed, SL-resolution is almost identical to model elimination plus the factoring rule (as defined in Section 4.8.4) and, hence, to regular connection tableaux with factoring.

5.3. Tableaux and (Disjunctive) Logic Programming

In order to benefit from this section, a little background in logic programming is of advantage, which can be gained from [Lloyd 1987].

Historically, one root of logic programming [Kowalski 1974] is SL-resolution [Kowalski and Kuehner 1971] and, hence, model elimination. It is not surprising, therefore, that logic programming and many of its extensions have natural interpretations as variants of connection tableaux.

More formally, let S be an unsatisfiable set of Horn clauses. Corollary 4.15 guarantees the existence of a connection tableau proof T for S starting with a *query* Q . Regularity is not enforced. By a trivial induction on the depth of T , using that S is Horn and connectedness of T , one obtains that the positive literals of T are exactly the literals at leaf nodes. As a consequence, no reduction steps occur in T and the connection literal of clauses used in extension steps is always the positive literal of a rule or a fact in S .

It was noted on page 128 that the existence of T is independent of the *select branch* strategy, so we may assume that to be left-first. In this case, T is a notational variant of a *Prolog-SLD resolution* refutation of $\{Q\} \cup (S - \{Q\})$ in the usual sense [Lloyd 1987]. This remains true on the first-order level.

5.5. EXAMPLE. To approximate the usual notation of logic programming, we employ rule notation of clauses as introduced in Section 4.6.¹⁹

$$S = \{C_1(x) = Q(x) \wedge R \rightarrow P(x), \text{ true} \rightarrow P(d), C_2(x) = S(x) \wedge T(x) \rightarrow Q(x), \\ \text{true} \rightarrow Q(a), \text{true} \rightarrow S(b), \text{true} \rightarrow S(c), \text{true} \rightarrow T(c), \text{true} \rightarrow R, \\ P(x) \rightarrow \text{false}\}$$

One of several possible connection tableaux for S starting with the only query $P(x) \rightarrow \text{false}$ in S is displayed on the left in Figure 22.

The open branches are the ones ending with $\neg T(z)\{z \mapsto c\} = \neg T(c)$ and $\neg R$, respectively. The tableau contains a lot of redundant information such as branches already closed and non-leaf nodes (which are never used, because only connected extension steps are made). Thus a less redundant notation for connected Horn clause tableaux merely records (a) the sequence of leaves of open branches and (b) the clause instance and MGU leading to the successor tableau. The result of this simplification is displayed in Figure 22 on the right: each node represents a connection tableau constructed from its parent node. The tableau on the left corresponds to the bottom-most node in black. The grey nodes complete the proof.

At the same time, the structure on the right constitutes a Prolog-SLD refutation of S from $P(x) \rightarrow \text{false}$ in common logic programming notation [Lloyd 1987]. Another valid point of view is to interpret the figure as a model elimination proof with left-first branch selection, where only the rightmost block of type B literals in a model elimination chain is represented.

In summary, although the Prolog procedure is often presented as a resolution refinement, it can be completely and naturally fitted into the connection tableau

¹⁹Without loss of generality, clause sets are assumed to be in *goal normal form* in this section, that is, they contain at most one query clause. When there is more than one query clause, this can easily be achieved by adding a new head predicate P to each query as well as the new query $P \rightarrow \text{false}$.

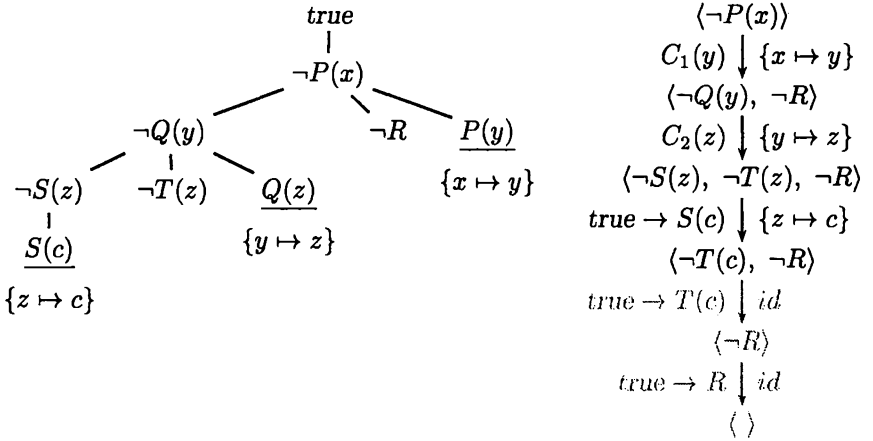


Figure 22: Illustration of Example 5.5.

(and hence: model elimination) framework. In the remainder of this section I intend to show that it is even advantageous to do so.

The following terminology comes handy: given a clause set S in rule notation, consider a clause tableau T for S . An occurrence of a literal L in T that was introduced in an extension step using rule $C \in S$ such that L occurs in the premise or body of C is called a *body literal*. Similarly, literal occurrences in T that stem from conclusions or heads of rules in S are called *head literal*. Further, in a reduction step in a clause tableau that involves L as the leaf literal, we say that the reduction *is from* L . Finally, the literal $L \in C$ in a (weakly) connected extension step using clause C in branch B such that \bar{L} occurs on B is referred to with the phrase *extension via* L .

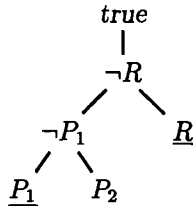
Disjunctive logic programming, instances of which are *Non-Horn clause logic programming* [Plaisted 1988] and *Near-Horn logic programming* [Loveland 1987], is the attempt to lift the Horn restriction from Prolog while retaining the efficient implementability of Prolog-SLD resolution. One line of development is to implement model elimination (that is: regular connection tableaux) for arbitrary clauses directly via an extended Prolog Abstract Machine [Letz et al. 1992]. This is discussed in detail in [Letz and Stenz 2001] (Chapter 28 of this Handbook). In a second group of papers one can find several suggestions how to extend Prolog-SLD resolution such that general clauses can be handled [Plaisted 1982, Gabbay and Reyle 1984, Plaisted 1988, Loveland 1991, Reed and Loveland 1992a, Baumgartner and Furbach 1994, Baumgartner and Furbach 1997, Baumgartner and Furbach 1998]. These calculi are in some aspects more restrictive, and in others more general than connection tableaux and model elimination.²⁰ We start our discussion by two examples that highlight some differences.

²⁰The proximity of near-Horn Prolog to model elimination was pointed out by Reed and Loveland [1992b] and again by Baumgartner and Furbach [1994].

5.6. EXAMPLE. Consider $S_1 = \{\text{true} \rightarrow P \vee Q, P \wedge Q \rightarrow \text{false}, P \rightarrow Q, Q \rightarrow P\}$. In the Prolog-SLD restriction of connection tableau no reduction steps are allowed. Obviously, no connection tableau proof without reduction steps can exist for S_1 : as there are no unit clauses, no extension step diminishes the number of open branches.

The example shows that reduction steps are necessary to handle general clause sets. The second example concerns positive leaf literals of tableau proofs. Recall that these cannot occur in the Horn case when one starts the proof with a query.

5.7. EXAMPLE. Consider $S_2 = \{P_1 \rightarrow R, P_2 \rightarrow R, R \rightarrow \text{false}, \text{true} \rightarrow P_1 \vee P_2\}$. Below is a partial connection tableau proof for S_2 :



In connection tableaux the open branch ending in P_2 can be extended with $P_2 \rightarrow R$ which completes the proof with one more extension using $R \rightarrow \text{false}$ (the case when $P_2 \rightarrow R$ is used before $P_1 \rightarrow R$, leads to a symmetric situation). To do so, however, is a violation of a main principle of Prolog-interpretation, namely, that the current subgoal (the leaf) may only be unified with a *head* literal of a rule. Indeed, in the linear format for Prolog-SLD resolution discussed in Example 5.5, the literal P_2 is not represented. It is, of course, possible to extend the linear proof format to accommodate head literals (see [Reed and Loveland 1992a] and Example 5.8), but I prefer to use the clause tableau format, where all required information is present and subgoal dependencies are easy to see. Some authors prefer to use a sequent-style format [Plaisted 1990, Reed and Loveland 1992b].

The principle of Prolog execution, not to extend via body literals, discussed in the previous example is often referred to as “no use of *contrapositives*”. The set of contrapositives of a clause $L_1 \vee \dots \vee L_n$ is the rule set

$$\{\overline{L_1} \wedge \dots \wedge \overline{L_{i-1}} \wedge \overline{L_{i+1}} \wedge \dots \wedge \overline{L_n} \rightarrow L_i \mid 1 \leq i \leq n\}.$$

Apart from implementational issues it is obvious that the possibility of extension steps with arbitrary contrapositives such as in connection tableaux and model elimination increases the local search space. Thus the interest in calculi that are complete without contrapositives.

For the rest of Section 5.3 we assume that clause sets S are given in rule notation and *no other contrapositives than the explicitly given rules can be used in tableau extension steps*.

How does one render connection tableaux without contrapositives complete? The situation is quite similar to tableaux with selection function discussed in Sec-

tion 4.5.2, because the head literals can be seen as the selected literals of a clause.²¹ It does not come as a surprise then, that the remedy is similar: certain *restart steps*, that is, extension steps without a connection are permitted to regain completeness.

A general framework of clause tableau calculi for disjunctive logic programming is, therefore, given by the following coordinates:

1. the first extension step is with a query, see Corollary 4.15;
2. connected extensions steps, generalizing the Prolog-SLD resolution rule, are permitted;
3. a list of permitted contrapositives of clauses is either explicitly given in rule notation or exactly the positive literals are head literals and the negative literals are body literals;
4. reduction steps (called *ancestor cancellation* rule by Reed and Loveland [1992b] who derive the name from ancestor resolution) are only allowed from body literals;
5. restart extension steps are only allowed below head literals.

The last two items constitute the parameterizable part. Disjunctive logic programming calculi are not proof confluent as the example $S = \{\text{true} \rightarrow C, C \rightarrow A, B \rightarrow A, A \rightarrow \text{false}\}$ shows: any tableau starting with the last clause, followed by a extension step with the second but last clause, cannot be extended, although S is clearly unsatisfiable. Hence, all calculi described in the remainder of this section are not proof confluent, unless otherwise noted.

All deduction procedures incorporating at least the first four ingredients are classified as the *ancestry family* of deductive procedures by Reed and Loveland [1992b]. In the remainder of the section I define some of the more important representatives within the tableau framework.

5.3.1. Near-Horn Prolog, Simplified Problem Reduction Format

Near-Horn Prolog, later called *unit near-Horn Prolog* (UnH-Prolog), was suggested in [Loveland 1987, Loveland 1991]. A variant called *inheritance near-Horn Prolog* (InH-Prolog) is due to [Loveland and Reed 1991]. The most detailed reference on near-Horn Prolog and related procedures is [Reed and Loveland 1992b].

In UnH-Prolog restart steps are allowed with query clauses and with any clause containing a head literal that is weakly connected to the current branch (when head literals are considered as selected literals this is exactly a weakly connected extension step of tableaux with selection function, see Section 4.5.2). In UnH-Prolog restarts can only occur below head literals and reduction steps are only allowed to the head literal above the most recent restart.

In InH-Prolog, on the other hand, restart steps are only permitted with query clauses, but reduction steps can be to the head literal above *any* restart step on the current branch.

Both, UnH- and InH-Prolog, use a left-first branch selection rule and both feature the so-called *cancellation pruning rule* which excludes certain redundant proofs:

²¹Tableaux with selection function select the *body* literals of a clause, but Prolog proceeds “top-down” from the query and extension steps are via head literals.

each head literal above a restart step must be used in a reduction step with no restart step in between.

5.8. EXAMPLE. As its name suggests, near-Horn Prologs are generalizations of Prolog-SLD resolution. We stress this by rewriting Example 5.7 in a Prolog-like notation [Reed and Loveland 1992b]. The separator '#' is followed by a list of active heads available for reduction: the head literals in the current tableau branch (only the most recently introduced head literal, in the case of UnH-Prolog). Next is the list of deferred subgoals (leaves of open branches), surrounded by curly brackets. The tableau in the Example 5.7 corresponds to the following derivation, which is both an UnH- and an InH-Prolog derivation:

$$\begin{array}{l} \text{?- false} \# \\ \text{?- } R \# \\ \text{?- } P_1 \# \\ \text{?-} \# \{P_2\} \end{array}$$

In UnH-Prolog one can continue with a restart step at R and the only deferred subgoal P_2 :

$$\begin{array}{l} \text{?- } R \# P_2 \\ \text{?- } P_2 \# P_2 \\ \text{?-} \# P_2 \end{array}$$

This time, clause $P_2 \rightarrow R$ is used, which makes reduction with P_2 possible. As there are no more deferred subgoals, the proof is finished.

In InH-Prolog only a restart step at *false* is possible after the first block, leading to a slightly longer proof.

It was observed by Baumgartner and Furbach [1994] that completeness of InH-Prolog entails completeness of weak connection tableaux (Section 4.3.2) and, hence, of the clausal version of the connection method (Section 3.5), even without using contrapositives of clauses provided that the input is in goal normal form. The reason is simply that the query clause of a restart step in an InH-Prolog proof is weakly connected to the root literal in this case. Therefore, an InH proof (which does not use contrapositives) is at the same time a weakly connected clause tableau proof.

In Plaisted's [1982] *simplified problem reduction format* (SPRF) reduction steps are as in InH-Prolog while restart steps may occur anytime with non-Horn clauses. Branch selection is flexible. In SPRF-D (for SPRF with delay) [Plaisted 1988] restart steps are restricted exactly as in InH-Prolog, but need not to occur below a head literal. Up to this feature and the cancellation pruning rule (not present in SPRF-D) both calculi are identical [Reed and Loveland 1992a].

Another deduction system related to InH-Prolog is N-Prolog [Gabbay and Reyle 1984, Gabbay 1985]. As N-Prolog is not clause-based and has an intuitionistic semantics, a comparison hinges on the translation chosen for clause representation. Reed and Loveland [1992a] showed that it is possible to embed both InH-Prolog and SPRF into N-Prolog.

SLWV-resolution [Pereira et al. 1992], often discussed in connection with near-Horn Prolog [Reed and Loveland 1992*b*, Baumgartner and Furbach 1994], was mentioned in Section 4.5.3, because it is actually an instance of tableaux with selection function.

5.3.2. Restart Model Elimination

While near-Horn Prolog and its relatives were developed as generalizations of Horn clause logic programming, the motivation of *restart model elimination* for Baumgartner and Furbach [1994] was to obtain a specialized version of connection tableaux/model elimination that is complete without contrapositives.

The base calculus of restart model elimination has exactly the same restart rule as InH-Prolog: only query clauses (in goal normal form: *the* query clause) can be used as a restart below head literals. Reduction is completely unrestricted, hence InH-Prolog is a refinement of restart model elimination.

Within the restart model elimination framework the restriction of reduction steps to be only from body literals employed in InH-Prolog is called *strict restart model elimination* in [Baumgartner and Furbach 1994]. This leaves the cancellation pruning rule and left-first branch selection strategy as the only remaining difference between InH-Prolog and strict restart model elimination. Left-first branch selection is standard in PTP technology-based implementations such as the one reported in [Baumgartner and Furbach 1994]. The gap is closed by [Baumgartner and Furbach 1998] where strict restart model elimination with cancellation pruning is considered.

Restart model elimination has a number of refinements [Baumgartner and Furbach 1994, Baumgartner and Furbach 1998] which justify the investigation of this procedure in its own right. The following refinements were investigated:

Head Selection Function: similar as in Section 4.5.2 a selection function is used to choose one head literal to be used in an extension step.

Blockwise Regularity: several restarts on the same branch with the same literal are admissible to maintain completeness, but in between subsequent restarts regularity can be enforced.

Independence of Goal Clause: clause sets in goal normal form have exactly one query clause, but in the second extension step any of the query clauses that existed before transformation to goal normal form can be used. Independence of the goal clause means that completeness is independent of the choice of the clause at the second extension step.

Not all refinements are compatible with each other, certain combinations result in an incomplete deduction system. Details are given in [Baumgartner and Furbach 1998].

An important issue when using Disjunctive Logic Programming as a programming language is the computation of all correct answer substitutions (closing substitutions for the tableau proof restricted to variables occurring in the query). Baumgartner, Furbach and Stolzenburg [1997] prove answer completeness of restart model elimination.

Many aspects discussed in the present section are covered in greater detail in [Reed and Loveland 1992*b*, Baumgartner and Furbach 1998].

5.3.3. Restart Tableaux

There are calculi that combine features of tableaux with selection function and of near-Horn Prolog/restart model elimination.

Restart tableaux [Pape and Hähnle 1997] can be motivated from the observation that the combination of selection functions and enforcing the weak connection condition for every extension step via a selected literal in general results in an incomplete calculus (Section 4.5.3). Adding restart steps in the sense of Section 4.5.2 gives completeness back. Restart tableaux restrict their restart steps to occur only in situations when no weakly connected extension step via a selected literal is possible, but, in contrast to near-Horn Prolog and restart model elimination, these restart steps may also occur below body literals. This relaxation makes restart tableaux a *proof confluent* calculus.

Restart tableaux admit to select an arbitrary subset S_c of the input clause set S such that only connected extension steps can be performed with clauses from S_c . Depending on the degree of “connectedness”, a more restrictive notion of regularity than in restart model elimination may be enforced. Thus restart tableaux combine features of restart model elimination and tableaux with selection function.

There is a non-proof confluent version of restart tableaux, called *strict restart tableaux* [Pape and Hähnle 1997], which does not allow restart steps below body literals, but has a more liberal regularity concept. It is a generalization of strict restart model elimination.

5.4. Davis-Putnam Procedure and related methods

Clause tableaux with the cut rule (4.10) are a redundant calculus in the sense that the cut rule is not required for completeness. It was noted in Section 4.8.5 that the presence of cut can blow up the search space considerably. Mondadori [1988] designed a tableau system with cut called *KE* for full first-order logic, where every rule, including the cut rule, is essential for completeness. Together with D’Agostino, KE was further developed and extended to non-classical logics in a series of papers [Mondadori 1989*a*, Mondadori 1989*b*, D’Agostino 1990, D’Agostino and Mondadori 1994].

KE in the clausal case is a close variant of weak connection tableaux (Section 4.3.2) and can be succinctly described as follows: take the atomic cut rule (4.10), the initialization rule of clause tableaux (Definition 4.1(i)), and these extension rules (compare with (ii’’) on page 131) are:

- (ii’’) Let T be a KE-tableau for S , B a branch of T containing L , $L_1 \vee \dots \vee L_r$ is on B or a new instance of $C \in S$. If \bar{L} , L_i (where $i \in \{1, \dots, r\}$) are unifiable with MGU σ and the tree T' is constructed by extending B with $\min\{2, r\}$ new subtrees, where the nodes of the new subtrees are L_i and, if

$r \geq 2$, $L_1 \vee \cdots \vee L_{i-1} \vee L_{i+1} \vee \cdots \vee L_r$, then $T'\sigma$ is a KE-tableau for S , in which the branch ending with $L_i\sigma$ is marked as closed. (This is called a *KE extension step*.)

(iii''') Let T be a tableau for S , B a branch of T , L a new instance of a unit clause in S . Then the tableau constructed from T by extending B with L is a KE-tableau for S . (This is called a *unit extension step*.)

Formally, the resulting calculus is not an instance of the clause tableau framework, but of non-clausal tableaux, because nodes may be clause instances (not just literals). Application of the extension rules of KE-tableaux does never increase the number of open branches, so the system is incomplete without the atomic cut rule, which is called *principle of bivalence* by Mondadori [1988].

Rule (ii''') is better known under the name *unit resolution*. Its propositional version originates in [Davis and Putnam 1960], where it is called *elimination of one-literal clauses*. The *affirmative-negative* rule in [Davis and Putnam 1960], better known as *pure clause rule* avoids all extensions with clauses that contain an unconnected literal; it is only partly realized in the weak connection condition. Finally, atomic cut is a notational variant of the *splitting rule* in [Davis et al. 1962].

To summarize, in the ground clause case, Mondadori's [1988] KE system and the *Davis-Putnam-Loveland-Logeman procedure* [Davis and Putnam 1960, Davis et al. 1962] are very similar and closely related to weak connection tableaux with atomic cut. The idea is, of course, to apply the splitting/cut rule only when no other rule is applicable. This heuristic is very useful for propositional logic, but it makes the Davis-Putnam-Loveland procedure problematic to lift, because splitting may be delayed indefinitely.

It is justified to view the full non-clausal KE system as a non-clausal version of the Davis-Putnam-Loveland procedure (up to the pure clause rule). Within the non-clausal tableau framework with *analytic cut* (the cut formula can be any subformula of the input), it is characterized by unchanged non-branching (type α , γ , δ) rules, while type β rules are used in the following version:

$$\frac{\frac{\beta}{\beta_i}}{\beta_1 \vee \cdots \vee \beta_{i-1} \vee \beta_{i+1} \vee \cdots \vee \beta_n} \quad (5.1)$$

Mondadori [1989b] also suggested a system called *KI* consisting of "inverted" tableau rules that build up complex formulas from simpler ones. At the propositional level they are:

$$\frac{\beta_i}{\beta} \quad \begin{array}{c} \alpha_1 \\ \vdots \\ \alpha_n \\ \alpha \end{array} \quad (5.2)$$

These rules are restricted to analytic use in the sense that their conclusion must occur as a subformula in the input. They may either be used in addition to the KE

rules (in which case they can be seen as an approximation of Massacci’s [1998*b*] simplification rule, see Section 3.5.3), or alternatively, they form yet another complete system together with *atomic cut*.

Another tableau calculus and relative of KE is *Stålmarck’s procedure* [Sheeran and Stålmarck 2000], which is underlying a commercially successful satisfiability checker for propositional logic, the *Prover Plug-In*. The calculus consists of (non-branching) KE-like rules for a restricted propositional language, namely, formulas of the form $P \leftrightarrow (Q \rightarrow R)$, where P, Q, R are atoms or logical constants. They are called *triplets* and form the tableau nodes. In addition, there is the so-called *dilemma rule*, a generalization of atomic cut: after applying the cut, each branch is saturated with respect to the non-branching rules; if neither branch is closed or gives a model, then both branches are recombined by computing the intersection of their triplets. The nesting depth of dilemma rule applications can be limited to a pre-set bound.

6. Comparing Calculi

In this section I suggest some coordinates along which the plethora of existing tableau-like calculi (and others) can be cartographed. It is kept short, because most of the material is not specific to tableau-like calculi.

A useful qualitative distinction are the properties *proof confluence* and *branching*. The latter encompasses calculi, where the derivation process branches or forks, see also [Bachmair and Ganzinger 2001] (Chapter 2 in this Handbook). In Table 4 are typical representatives of each kind of calculus.

	branching	non-branching
proof confluent	tableaux with unification	Robinson resolution
not proof confluent	connection tableaux	Prolog SLD-resolution

Table 4: Qualitative classification of calculi.

Calculi that are not proof confluent cannot be used to find models of satisfiable formulas. They make no sense to use with propositional formulas either, because backtracking is expensive and can easily be avoided in the propositional case. Implementation of branching calculi demands more general data structures than of non-branching ones. In practice, implementation-oriented branching calculi are often formulated sequentially, for example, model elimination. Another property that severely affects the choice implementation techniques is destructiveness. For example, it is not straightforward to build strong redundancy criteria such as subsumption or regularity into destructive calculi, see Section 4.4.

Deeper insight into similarities and differences among calculi is often gained by presenting them within a unifying theoretical framework, such as the tableau framework of the present chapter. The point of view of matrices is taken in

[Bibel 1982*b*, Bibel and Eder 1992]; *consolution* combines aspects from resolution and the connection method and can be used to analyze a broad class of calculi [Eder 1992, Baumgartner and Furbach 1993].

Another objective criterion for evaluation of calculi is provided by an investigation into their complexity.

First of all, one can determine the complexity of the decision problem of various subtasks associated with (tableau) proof search. It is well-known that clause subsumption is an NP-complete problem [Garey and Johnson 1979]; the same is true, for example, for computing hyper tableau extension steps in first-order logic (see Section 4.6.2) or for solving constraints arising in ordered tableaux (see Section 4.5.4) from certain literal orders [Pape 1996].

A more basic problem is encountered by the realization that tableau-like methods provide a constructive proof of Herbrand's Theorem 2.3. A tableau T can be considered as a formula ϕ_T , where branches form conjunctions over their literals and ϕ_T is a disjunction over the branches of T . Let x_1, \dots, x_r be the free variables of ϕ_T . By tableau soundness (Theorem 3.12), T can be closed iff $(\forall x_1, \dots, x_r)\phi_T$ is unsatisfiable and has Herbrand complexity $m = 1$. The problem, whether the sentence in Herbrand's Theorem is unsatisfiable with multiplicity $m = 1$, is called the *formula instantiation problem* in [Voronkov 1998]; it is shown to be decidable and Σ_2^P -complete, if the signature of ϕ_T has at least two symbols.

A completion mode used in tableau-like proof procedures (see Section 3.3) can be seen as a strategy to approximate the required multiplicity of a formula to show its unsatisfiability with Herbrand's Theorem. In [Voronkov 1998] such strategies are discussed and analyzed in an abstract framework. All of the mentioned problems become substantially more complex, if part of the formula signature is interpreted relative to theories (such as equality), see [Degtyarev and Voronkov 2001*a*] (Chapter 10 in this Handbook).

It is important to know the complexity of each part of a proof search procedure, but one would also like to have results about the relative overall performance of various refinements. One criterion that received much attention is *minimal proof length*, the size of a minimal proof for a formula ϕ , based on a suitable notion of proof length. Our definition of tableau size is an adequate measure. For propositional resolution, one can take the sum of the sizes of generated clauses (it was shown in [Letz 1993] that there can be no adequate measure for first-order resolution). Cook and Reckhow [1979] suggested to compare calculi based on the relative length of minimal proofs:

6.1. DEFINITION. Let \mathcal{P} and \mathcal{Q} be sound and complete calculi. \mathcal{P} can *p-simulate* \mathcal{Q} iff there is a polynomial p such that for all formula sets Φ : if there is a proof for Φ in \mathcal{Q} of length n , then there is a proof of Φ in \mathcal{P} of at most length $p(n)$. \mathcal{P} and \mathcal{Q} are *p-equivalent* iff they can p-simulate each other.

There is a large number of results on p-simulation for tableau and other calculi, for example, [Eder 1992, Letz 1993]. Because of technical difficulties, the vast majority are for the propositional, clausal case, but there are exceptions such as

[Baaz and Fermüller 1995, Egly 1997] showing that non-elementary differences in complexity exist on the first-order level for even closely related calculi. There are some surprises, for example, clause tableau cannot p-simulate truth table checking (see Example 4.33 and [D'Agostino 1992]) or n -ary type β tableau rules cannot p-simulate binary ones (see Section 3.2.6 and [Massacci 1998a]). Regular connection tableaux (and, hence model elimination) are pretty weak calculi in this hierarchy. For a number of reasons, relative proof length complexity is of limited use to predict the practical behaviour of calculi:

- relative proof length complexity measures the *nondeterministic* power of calculi, nothing is said about how long it takes to *find* a proof; for this, the search space size would be a more suitable measure;
- the stronger the nondeterministic power of a calculus, the larger is often the search space (witness tableaux with lemmas, Section 4.8.5);
- the notion of p-simulability is often too coarse, for example, all variants of tableaux with lemmas are p-equivalent, regardless of regularity or connectedness;
- the behaviour of calculi on the first-order level can easily override any differences on the propositional level.

An attempt to formalize the search space associated with calculi is made by Plaisted and Zhu [1997]. The search space is formalized as a directed graph whose nodes represent states in a proof and arcs define reachability among these states. Plaisted and Zhu [1997] analyze width, depth and size of nodes of search space graphs for a number of calculi and propositional Horn clauses. Satisfiability of the latter is well known to be decidable in linear time [Dowling and Gallier 1984], but many calculi turn out to be exponential along at least one dimension of their search space. As the Horn fragment is practically important, one can argue that the performance of a calculus there is significant also for its overall behaviour to a certain extent.

Apart from theoretical investigations one can and does perform experimental evaluation based on concrete implementations. There is an extensive problem collection (the TPTP library) for first-order theorem provers [Suttner and Sutcliffe 1999] and an annual competition [Sutcliffe and Suttner 1999]. This is a good way to test correctness and base speed of the implementation of an automatic theorem prover, but it does not necessarily tell everything on the usability of a system. The latter is perhaps best evaluated within larger case studies or applications. It is unlikely that the same deduction paradigm suits all needs. In practice, often a combination of several techniques is successful: see [Weidenbach 2001] (Chapter 27) of this Handbook for a point in case. There is also indication that the usefulness of fully automated systems is limited, whereas they have a large potential in connection with interactive systems [Joyce and Seger 1993, Ahrendt et al. 1998, Dahn and Schumann 1998].

7. Historical Remarks & Resources

I can only give a very rough sketch here. More detailed accounts of the history of tableau-like theorem proving can be found in [Bibel and Schmitt 1998, pp. 4–7] and [Fitting 1999].

Despite their recent flourishing, the history of tableau methods is much older than that of resolution. They can be traced back to the cut-free version of Gentzen's [1935] sequent calculus. Hintikka [1955] and Beth [1955] abstracted from structural rules in sequent calculi (essentially treating sequents as sets of formulas), improved the proof representation, and introduced signed formulas. They also stressed the semantic view of tableaux as a procedure that tries systematically to find a counter example for a given formula (a model in which its negation is true) as opposed to Gentzen's purely proof theoretical motivation. Further improvements were made by Schütte [1960]; Smullyan's [1995] elegant formulation, employing unifying notation which greatly simplified matters, became very popular while similar contributions by Lis [1960] unfortunately went unnoticed.

Many important improvements directed to automated proof search in sequent/tableau/model elimination calculi were made quite early: free variables [Kanger 1957, Prawitz 1960], unification [Loveland 1969, Bibel 1982b, Andrews 1981], proof representation and connection refinements [Davydov 1973, Bibel and Schreiber 1975, Andrews 1976].

The relative success of resolution-based theorem proving, however, eclipsed this progress and serious implementations of tableau-like calculi are spurious before the late 1980s [Brown 1978, Oppacher and Suen 1986].

In the last decade tableau-like calculi became focal points of research again, spurred by the success of efficient implementations and the demand for a computational treatment of non-classical logics. The tableau community gathers in an international conference,²² where many of the relevant results are now published. Part I of [Bibel and Schmitt 1998] contains survey articles on various aspects of state-of-the-art research on tableau methods. The Handbook of Tableau Methods [D'Agostino, Gabbay, Hähnle and Posegga 1999] contains extensive information, not limited to automated reasoning.

Acknowledgments

Uli Furbach and Philippe de Groote served as second readers of this chapter. Their careful reading and constructive criticism resulted in numerous large and small improvements. In addition, Andrei Voronkov proofread most of this chapter and provided many corrections and useful suggestions. Thanks are also due to Bernhard Beckert, Jean Goubault-Larrecq, Ryuzo Hasegawa, and Reinhold Letz for discussions, helpful observations, and suggestions. Reinhold Letz contributed Example 4.9. Jeff Lansing corrected a number of errors in my English.

²²[112www.ira.uka.de/TABLEAUX/](http://www.ira.uka.de/TABLEAUX/)

Bibliography

- AHRENDT W., BECKERT B., HÄHNLE R., MENZEL W., REIF W., SCHELLHORN G. AND SCHMITT P. H. [1998], Integration of automated and interactive theorem proving, in W. Bibel and P. Schmitt, eds, 'Automated Deduction: A Basis for Applications', Vol. II, Kluwer, chapter 4, pp. 97–116.
- ANDREWS P. [1971], 'Resolution in type theory', *Journal of Symbolic Logic* **36**, 414–432.
- ANDREWS P. [1976], 'Refutations by matings', *IEEE Transactions on Computers* **C-25**(8), 801–807.
- ANDREWS P. [1981], 'Theorem proving through general matings', *JACM* **28**, 193–214.
- AVRON A. [1993], 'Gentzen-type systems, resolution and tableaux', *Journal of Automated Reasoning* **10**(2), 265–281.
- BAADER F. AND SNYDER W. [2001], Unification theory, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 8, pp. 445–532.
- BAAZ M., EGLY U. AND LEITSCH A. [2001], Normal form transformations, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 5, pp. 273–333.
- BAAZ M. AND FERMÜLLER C. G. [1995], Nonelementary speedups between different versions of tableaux, in P. Baumgartner, R. Hähnle and J. Posegga, eds, 'Proc. 4th Workshop on Deduction with Tableaux and Related Methods, St. Goar, Germany', Vol. 918 of *LNCS*, Springer-Verlag, pp. 217–230.
- BAAZ M., FERMÜLLER C. AND SALZER G. [2001], Automated deduction for many-valued logics, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 20, pp. 1355–1402.
- BACHMAIR L. AND GANZINGER H. [2001], Resolution theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19–99.
- BAUMGARTNER P. [1998], Hyper Tableaux — The Next Generation, in H. de Swart, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands', number 1397 in 'LNCS', Springer-Verlag, pp. 60–76.
- BAUMGARTNER P., EISINGER N. AND FURBACH U. [1999], A confluent connection calculus, in H. Ganzinger, ed., 'Proc. 16th International Conference on Automated Deduction, CADE-16, Trento, Italy', Vol. 1632 of *LNCS*, Springer-Verlag, pp. 329–343.
- BAUMGARTNER P., EISINGER N. AND FURBACH U. [2000], A confluent connection calculus, in S. Hölldobler, ed., 'Intellectics and Computational Logic — Papers in Honor of Wolfgang Bibel', Kluwer.
- BAUMGARTNER P. AND FURBACH U. [1993], 'Consolution as a framework for comparing calculi', *Journal of Symbolic Computation* **16**, 445–477.
- BAUMGARTNER P. AND FURBACH U. [1994], 'Model Elimination without Contrapositives and its Application to PTP', *Journal of Automated Reasoning* **13**, 339–359.
- BAUMGARTNER P. AND FURBACH U. [1997], Refinements for Restart Model Elimination, in 'Proc. International Workshop on First Order Theorem Proving', Technical Report, RISC-Linz.
- BAUMGARTNER P. AND FURBACH U. [1998], Variants of clausal tableaux, in W. Bibel and P. Schmitt, eds, 'Automated Deduction: A Basis for Applications', Vol. I, Kluwer, chapter 3, pp. 73–101.
- BAUMGARTNER P., FURBACH U. AND NIEMELÄ I. [1996], Hyper tableaux, in J. J. Alferes, L. M. Pereira and E. Orłowska, eds, 'Proc. European Workshop: Logics in Artificial Intelligence, JELIA', Vol. 1126 of *LNCS*, Springer-Verlag, pp. 1–17.
- BAUMGARTNER P., FURBACH U. AND STOLZENBURG F. [1997], 'Computing answers with model elimination', *Artificial Intelligence Journal* **90**(1–2), 135–176.
- BECKERT B. [1998], Integrating and Unifying Methods of Tableau-based Theorem Proving, PhD thesis, University of Karlsruhe, Department of Computer Science.

- BECKERT B. [2000], Depth-first proof search without backtracking for free-variable clausal tableaux, in P. Baumgartner and H. Zhang, eds, 'Proc. Third Int. Workshop on First-Order Theorem Proving (FTP), St. Andrews, Scotland', Fachberichte Informatik 5/2000, University of Koblenz, Institute for Computer Science, pp. 44–55.
- BECKERT B. AND GORÉ R. [1997], Free variable tableaux for propositional modal logics, in 'Proc. International Conference on Theorem Proving with Analytic Tableaux and Related Methods, Pont-a-Mousson, France', Vol. 1227 of *LNCS*, Springer-Verlag, pp. 91–106.
- BECKERT B. AND HÄHNLE R. [1998], Analytic tableaux, in W. Bibel and P. Schmitt, eds, 'Automated Deduction: A Basis for Applications', Vol. I, Kluwer, chapter 1, pp. 11–41.
- BECKERT B., HÄHNLE R., OEL P. AND SULZMANN M. [1996], The tableau-based theorem prover *3TAP*, version 4.0, in M. McRobbie and J. Slaney, eds, 'Proc. 13th Conference on Automated Deduction, New Brunswick/NJ, USA', Vol. 1104 of *LNCS*, Springer-Verlag, pp. 303–307.
- BECKERT B., HÄHNLE R. AND SCHMITT P. H. [1993], The *even more* liberalized δ -rule in free variable semantic tableaux, in G. Gottlob, A. Leitsch and D. Mundici, eds, 'Proceedings of the third Kurt Gödel Colloquium KGC'93, Brno, Czech Republic', Vol. 713 of *LNCS*, Springer-Verlag, pp. 108–119.
- BECKERT B. AND POSEGGA J. [1995], 'lean^{TAP}: Lean tableau-based deduction', *Journal of Automated Reasoning* **15**(3), 339–358.
- BERKA, K. AND KREISER, L., EDS [1986], *Logik-Texte. Kommentierte Auswahl zur Geschichte der modernen Logik*, Akademie-Verlag, Berlin.
- BETH E. W. [1955], 'Semantic entailment and formal derivability', *Mededelingen van de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde, N.R.* **18**(13), 309–342. Partially reprinted in [Berka and Kreiser 1986].
- BIBEL W. [1979], 'Tautology testing with a generalized matrix method', *Theoretical Computer Science* **8**, 31–44.
- BIBEL W. [1981], 'On matrices with connections', *JACM* **28**, 633–645.
- BIBEL W. [1982a], *Automated Theorem Proving*, Vieweg, Braunschweig.
- BIBEL W. [1982b], 'A comparative study of several proof procedures', *Artificial Intelligence* **18**(3), 269–293.
- BIBEL W. [1982c], Computationally improved versions of herbrand's theorem, in 'Logic Colloquium '81', North-Holland, pp. 11–28.
- BIBEL W. [1987], *Automated Theorem Proving*, second revised edn, Vieweg, Braunschweig.
- BIBEL W., BRÜNING S., EGLY U., KORN D. AND RATH T. [1995], Issues in theorem proving based on the connection method, in P. Baumgartner, R. Hähnle and J. Posegga, eds, 'Proceedings of the 4th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods', Vol. 918 of *LNCS*, Springer-Verlag, pp. 1–16.
- BIBEL W. AND EDER E. [1992], Methods and calculi for deduction, in D. M. Gabbay, C. J. Hogger and J. A. Robinson, eds, 'Handbook of Logic in Artificial Intelligence and Logic Programming', Vol. 1: Logical Foundations, Oxford University Press, pp. 67–182.
- BIBEL, W. AND SCHMITT, P., EDS [1998], *Automated Deduction: A Basis for Applications*, Kluwer.
- BIBEL W. AND SCHREIBER J. [1975], Proof search in a Gentzen-like system of first-order logic, in F. Gelenbe and F. Poitier, eds, 'International Computing Symposium', North-Holland, pp. 205–212.
- BROWN F. M. [1978], 'Towards the automation of set theory and its logic', *Artificial Intelligence* **10**(3), 281–316.
- BRY F. AND TORGE S. [1998], A deduction method complete for refutation and finite satisfiability, in J. Dix, L. F. del Cerro and U. Furbach, eds, 'Proc. 6th European Workshop on Logics in AI (JELIA)', Vol. 1489 of *LNCS*, Springer-Verlag, pp. 122–136.
- BRY F. AND YAHYA A. [1996], Minimal model generation with positive unit hyper-resolution tableaux, in P. Miglioli, U. Moscato, D. Mundici and M. Ornaghi, eds, 'Theorem Proving

- with Tableaux and Related Methods, 5th International Workshop, TABLEAUX'96, Terrasini, Palermo, Italy', Vol. 1071 of *LNCS*, Springer-Verlag, pp. 143–159.
- BRYANT R. E. [1986], 'Graph-based algorithms for Boolean function manipulation', *IEEE Transactions on Computers* **C-35**, 677–691.
- CANTONE D. AND NICOLOSI ASMUNDO M. [1998], A further and effective liberalization of the delta-rule in free variable semantic tableaux, in R. Caferra and G. Salzer, eds, 'Proc. Second Int. Workshop on First-Order Theorem Proving, FTP'98', Technical Report E1852-GS-981, Technische Universität, Wien (Austria), pp. 86–96.
- COOK S. AND RECKHOW R. [1979], 'The relative efficiency of propositional proof systems', *Journal of Symbolic Logic* **44**, 36ff.
- D'AGOSTINO M. [1990], Investigations into the Complexity of some Propositional Calculi, PhD thesis, Oxford University Computing Laboratory, Programming Research Group. Also Technical Monograph PRG-88, Oxford University Computing Laboratory.
- D'AGOSTINO M. [1992], 'Are tableaux an improvement on truth tables? Cut-free proofs and bivalence', *Journal of Logic, Language and Information* **1**, 235–252.
- D'AGOSTINO, M., GABBAY, D., HÄHNLE, R. AND POSEGGGA, J., EDS [1999], *Handbook of Tableau Methods*, Kluwer, Dordrecht.
- D'AGOSTINO M. AND MONDADORI M. [1994], 'The taming of the cut', *Journal of Logic and Computation* **4**(3), 285–319.
- DAHN I. AND SCHUMANN J. [1998], Using automated theorem provers in verification of protocols, in W. Bibel and P. Schmitt, eds, 'Automated Deduction: A Basis for Applications', Vol. III, Kluwer, chapter 8, pp. 195–224.
- DAVIS M., LOGEMANN G. AND LOVELAND D. [1962], 'A machine program for theorem-proving', *Communications of the ACM* **5**, 394–397.
- DAVIS M. AND PUTNAM H. [1960], 'A computing procedure for quantification theory', *JACM* **7**(3), 201–215.
- DAVYDOV G. V. [1973], 'Synthesis of the resolution method with the inverse method', *Journal of Soviet Mathematics* **1**, 12–18. Translated from Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta im. V. A. Steklova Akademii Nauk SSSR, vol. 20, pp. 24–35, 1971.
- DEGTAREV A. AND VORONKOV A. [1998], 'What you always wanted to know about rigid E-unification', *Journal of Automated Reasoning* **20**(1), 47–80.
- DEGTAREV A. AND VORONKOV A. [2001], Equality reasoning in sequent-based calculi, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 10, pp. 609–704.
- DOWLING W. AND GALLIER J. [1984], 'Linear-time algorithms for testing the satisfiability of propositional Horn formulæ', *Journal of Logic Programming* **3**, 267–284.
- EDER E. [1992], *Relative Complexities of First-Order Calculi*, Artificial Intelligence, Vieweg Verlag.
- EGLY U. [1997], Non-elementary speed-ups in proof length by different variants of classical analytic calculi, in D. Galmiche, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods', Vol. 1227 of *LNCS*, Springer-Verlag, pp. 158–172.
- FITTING M. [1999], Introduction, in M. D'Agostino, D. Gabbay, R. Hähnle and J. Posegga, eds, 'Handbook of Tableau Methods', Kluwer, Dordrecht, pp. 1–43.
- FITTING M. C. [1990], *First-Order Logic and Automated Theorem Proving*, Springer-Verlag, New York.
- FITTING M. C. [1996], *First-Order Logic and Automated Theorem Proving*, second edn, Springer-Verlag, New York.
- FUJITA H. AND HASEGAWA R. [1991], A model generation theorem prover in KLI using a ramified-stack algorithm, in K. Furukawa, ed., 'Proceedings 8th International Conference on Logic Programming, Paris/France', MIT Press, pp. 535–548.

- GABBAY D. M. [1985], 'N-Prolog: An extension of Prolog with hypothetical implication II—logical foundations, and negation as failure', *Journal of Logic Programming* 2(4), 251–283.
- GABBAY D. M. AND REYLE U. [1984], 'N-Prolog: An extension of Prolog with hypothetical implications I', *Journal of Logic Programming* 1(4), 319–355.
- GALLO G. AND URBANI G. [1989], 'Algorithms for testing the satisfiability of propositional formulae', *Journal of Logic Programming* 7(1), 45–62.
- GAREY M. R. AND JOHNSON D. S. [1979], *Computers and Intractability*, Freeman, San Francisco.
- GENTZEN G. [1935], 'Untersuchungen über das Logische Schliessen', *Mathematische Zeitschrift* 39, 176–210, 405–431. English translation [Szabo 1969].
- GIESE M. [2000], Proof search without backtracking using instance streams, position paper, in P. Baumgartner and H. Zhang, eds, 'Proc. Third Int. Workshop on First-Order Theorem Proving, St. Andrews, Scotland', Fachberichte Informatik 5/2000, University of Koblenz, Institute for Computer Science, pp. 227–228.
- GIESE M. AND AHRENDT W. [1998], Hilbert's ϵ -terms in automated theorem proving, in N. V. Murray, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Saratoga Springs/NY, USA', number 1617 in 'LNCS', Springer-Verlag, pp. 171–185.
- HÄHNLE R. [1999], Tableaux for many-valued logics, in M. D'Agostino, D. Gabbay, R. Hähnle and J. Posegga, eds, 'Handbook of Tableau Methods', Kluwer, Dordrecht, pp. 529–580.
- HÄHNLE R. AND KLINGENBECK S. [1996], 'A-ordered tableaux', *Journal of Logic and Computation* 6(6), 819–834.
- HÄHNLE R., MURRAY N. AND ROSENTHAL E. [1997], Completeness for linear regular negation normal form inference systems, in Z. W. Raś and A. Skowron, eds, 'Foundations of Intelligent Systems, 10th International Symposium, ISMIS'97, Charlotte, North Carolina, USA', number 1325 in 'LNCS', Springer-Verlag, pp. 590–599.
- HÄHNLE R. AND PAPE C. [1997], Ordered tableaux: Extensions and applications, in D. Galmiche, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France', Vol. 1227 of *LNCS*, Springer-Verlag, pp. 173–187.
- HASEGAWA R., FUJITA H. AND KOSHIMURA M. [1997], MGTP: a model generation theorem prover—its advanced features and applications, in D. Galmiche, ed., 'Proc. Int. Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France', Vol. 1227 of *LNCS*, Springer-Verlag, pp. 1–15.
- HASEGAWA R., INOUE K., OHTA Y. AND KOSHIMURA M. [1997], Non-Horn magic sets to incorporate top-down inference into bottom-up theorem proving, in W. McCune, ed., 'Proc. 14th International Conference on Automated deduction', Vol. 1249 of *LNCS*, Springer-Verlag, pp. 176–190.
- HASEGAWA R., KOSHIMURA M. AND FUJITA H. [1992], MGTP: A parallel theorem prover based on lazy model generation, in D. Kapur, ed., 'Proc. 11th International Conference on Automated Deduction', LNAI 607, Springer-Verlag, pp. 776–780.
- HERBRAND J. [1930], *Recherches sur la théorie de la démonstration*, Thèse de doctorat, Université de Paris, France. Reprinted in [Herbrand 1971].
- HERBRAND J. [1971], *Jacques Herbrand: Logical Writings*, edited by W. Goldfarb, Reidel, Dordrecht.
- HINTIKKA K. J. J. [1955], 'Form and content in quantification theory', *Acta Philosophica Fennica* 8, 7–55.
- JOYCE J. J. AND SEGER C.-J. H. [1993], Linking BDD-based symbolic evaluation to interactive theorem-proving, Technical Report TR-93-18, UBC.
- KANGER S. [1957], *Provability in Logic*, Vol. 1 of *Acta Universitatis Stockholmiensis*, Almqvist & Wiksell, Stockholm.

- KLINGENBECK S. AND HÄHNLE R. [1994], Semantic tableaux with ordering restrictions, in A. Bundy, ed., 'Proc. 12th Conference on Automated Deduction CADE, Nancy/France', Vol. 814 of *LNCS*, Springer-Verlag, pp. 708–722.
- KORF R. E. [1985], 'Depth-first iterative deepening: an optimal admissible tree search', *Artificial Intelligence* 27, 97–109.
- KOSHIMURA M. AND HASEGAWA R. [1999], A proof of completeness for non-Horn magic sets and its application to proof condensation, in N. Murray, ed., 'Position Papers, International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Saratoga Springs, NY, USA', Technical Report 99-1, Institute for Programming and Logics, Department of Computer Science, University at Albany – SUNY, pp. 101–116.
- KOWALSKI R. [1974], 'Predicate logic as a programming language', *Information Processing* 74, 569–574.
- KOWALSKI R. AND KUEHNER D. [1971], 'Linear resolution with selection function', *Artificial Intelligence* 2(3), 227–260.
- LETZ R. [1993], First-Order Calculi and Proof Procedures for Automated Deduction, PhD thesis, TH Darmstadt.
- LETZ R. [1998], Using matings for pruning connection tableaux, in C. Kirchner and H. Kirchner, eds, 'Proc. 15th International Conference on Automated Deduction (CADE), Lindau', Vol. 1421 of *LNCS*, Springer-Verlag, pp. 381–396.
- LETZ R., MAYR K. AND GOLLER C. [1994], 'Controlled integration of the cut rule into connection tableau calculi', *Journal of Automated Reasoning*, 13(3), 297–338.
- LETZ R., SCHUMANN J., BAYERL S. AND BIBEL W. [1992], 'SETHEO: A high-performance theorem prover', *Journal of Automated Reasoning* 8(2), 183–212.
- LETZ R. AND STENZ G. [2001], Model elimination and connection tableau procedures, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 28, pp. 2015–2114.
- LIS Z. [1960], 'Wynikanie semantyczne a wynikanie formalne (logical consequence, semantic and formal)', *Studia Logica* 10, 39–60. Polish, with Russian and English abstracts.
- LLOYD J. W. [1987], *Foundations of Logic Programming*, Second edn, Springer, Berlin.
- LOVELAND D. W. [1968a], A linear format for resolution, in 'Proc. IRIA Symp. Automatic Demonstration', Springer-Verlag, Versailles, France, pp. 147–162. Reprinted in [Siekmann and Wrightson 1983a].
- LOVELAND D. W. [1968b], 'Mechanical theorem proving by model elimination', *Journal of the ACM* 15(2), 236–251. Reprinted in: [Siekmann and Wrightson 1983a].
- LOVELAND D. W. [1969], 'A simplified format for the model elimination procedure', *Journal of the ACM* 16(3), 349–363. Reprinted in: [Siekmann and Wrightson 1983a].
- LOVELAND D. W. [1972], 'A unifying view of some linear Herbrand procedures', *Journal of the ACM* 19(2), 366–384.
- LOVELAND D. W. [1978], *Automated Theorem Proving. A Logical Basis*, Vol. 6 of *Fundamental Studies in Computer Science*, North-Holland.
- LOVELAND D. W. [1987], Near-Horn PROLOG, in J.-L. Lassez, ed., 'Proc. Fourth International Conference on Logic Programming, ICLP', MIT Press, Melbourne, Australia, pp. 456–469.
- LOVELAND D. W. [1991], 'Near-Horn Prolog and beyond', *Journal of Automated Reasoning* 7, 1–26.
- LOVELAND D. W. AND REED D. W. [1991], A near-Horn Prolog for compilation, in J.-L. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of Alan Robinson', MIT Press, Cambridge, MA.
- LOVELAND D. W., REED D. W. AND WILSON D. S. [1995], 'SATCHMORE: SATCHMO with RElevancy', *Journal of Automated Reasoning* 14(2), 325–351.
- LUCKHAM D. [1968], Refinements in resolution theory, in 'Proc. IRIA Symp. Automatic Demonstration', Springer-Verlag, Versailles, France, pp. 163–190. Reprinted in [Siekmann and Wrightson 1983a].

- MANTHEY R. AND BRY F. [1988], SATCHMO: A theorem prover implemented in Prolog, in E. Lusk and R. Overbeek, eds, 'Proceedings 9th Conference on Automated Deduction', LNCS, New York, Springer-Verlag, pp. 415–434.
- MASSACCI F. [1998a], Cook and Reckhow are wrong: subexponential proofs for their families of formulae, in H. Prade, ed., 'Proc. 13th European Conference on Artificial Intelligence, Brighton', John Wiley & Sons, pp. 408–409.
- MASSACCI F. [1998b], Simplification: A general constraint propagation technique for propositional and modal tableaux, in H. de Swart, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands', Vol. 1397 of LNCS, Springer-Verlag, pp. 217–232.
- MONDADORI M. [1988], Classical analytical deduction, Annali dell' Università di Ferrara, Nuova Serie, sezione III, Filosofia, discussion paper, n. 1, Università degli Studi di Ferrara.
- MONDADORI M. [1989a], Classical analytical deduction, part II, Annali dell' Università di Ferrara, Nuova Serie, sezione III, Filosofia, discussion paper, n. 5, Università degli Studi di Ferrara.
- MONDADORI M. [1989b], An improvement of Jeffrey's deductive trees, Annali dell' Università di Ferrara, Nuova Serie, sezione III, Filosofia, discussion paper, n. 7, Università degli Studi di Ferrara.
- MOSER M., IBENS O., LETZ R., STEINBACH J., GOLLER C., SCHUMANN J. AND MAYR K. [1997], 'SETHEO and E-SETHEO—the CADE-13 systems', *Journal of Automated Reasoning* 18(2), 237–246.
- NONNENGART A., ROCK G. AND WEIDENBACH C. [1998], On generating small clause normal forms, in H. Kirchner and C. Kirchner, eds, 'Proc. 15th International Conference on Automated Deduction, Lindau, Germany', number 1421 in 'LNCS', Springer-Verlag, pp. 397–411.
- NONNENGART A. AND WEIDENBACH C. [2001], Computing small clause normal forms, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 6, pp. 335–367.
- OHYA Y., INOUE K. AND HASEGAWA R. [1998], On the relationship between non-Horn magic sets and relevancy testing, in C. Kirchner and H. Kirchner, eds, 'Proc. 15th International Conference on Automated Deduction (CADE), Lindau', Vol. 1421 of LNCS, Springer-Verlag, pp. 333–347.
- OPPACHER F. AND SUEN E. [1986], Controlling deduction with proof condensation and heuristics, in J. H. Siekmann, ed., 'Proc. 8th International Conference on Automated Deduction', pp. 384–393.
- OPPACHER F. AND SUEN E. [1988], 'HARP: A tableau-based theorem prover', *Journal of Automated Reasoning* 4, 69–100.
- PAPE C. [1996], Vergleich und Analyse von Ordnungseinschränkungen für freie Variablen Tableau (in German), Interner Bericht 30/96, Universität Karlsruhe, Fakultät für Informatik.
- PAPE C. AND HÄHNLE R. [1997], Restart tableaux with selection function, in G. Gottlob, A. Leitsch and D. Mundici, eds, 'Fifth Kurt-Gödel-Colloquium, KGC'97, Vienna', Vol. 1289 of LNCS, Springer-Verlag, pp. 219–232.
- PEREIRA L. M., CAIRES L. AND ALFERES J. [1992], SLWV — a theorem prover for logic programming, in E. Lamma and P. Mello, eds, 'Proc. Third International Workshop on Extensions of Logic Programming, Bologna', Vol. 660 of LNCS, Springer-Verlag, pp. 1–23.
- PLAISTED D. A. [1982], 'A simplified problem reduction format', *Artificial Intelligence* 18, 227–261.
- PLAISTED D. A. [1988], 'Non-Horn clause logic programming without contrapositives', *Journal of Automated Reasoning* 4, 287–325.
- PLAISTED D. A. [1990], 'A sequent-style model elimination strategy and a positive refinement', *Journal of Automated Reasoning* 6, 389–402.
- PLAISTED D. A. AND GREENBAUM S. [1986], 'A structure-preserving clause form translation', *Journal of Symbolic Computation* 2, 293–304.

- PLAISTED D. A. AND ZHU Y. [1997], *The Efficiency of Theorem Proving Strategies: A Comparative and Asymptotic Analysis*, Vieweg Verlag, Braunschweig.
- POSEGGA J. [1993], Deduktion mit Shannongraphen für Prädikatenlogik erster Stufe (in German), PhD thesis, University of Karlsruhe. Diski 51, infix Verlag.
- POSEGGA J. AND SCHMITT P. H. [1995], 'Deduction with first-order Shannon graphs', *Journal of Logic and Computation* 5(6), 697-729.
- PRAWITZ D. [1960], 'An improved proof procedure', *Theoria* 26, 102-139. Reprinted in [Siekmann and Wrightson 1983b].
- PREISS R. [1998], Beweisvisualisierung und -analyse mit Hypergraphen, PhD thesis, Universität Karlsruhe, Fakultät für Informatik. Published by Shaker-Verlag, Aachen.
- RAGO G. [1994], Optimization, Hypergraphs and Logical Inference, PhD thesis, Dipartimento di Informatica, Università di Pisa. Available as Tech Report TD-4/94.
- REED D. W. AND LOVELAND D. W. [1992a], 'A comparison of three Prolog extensions', *Journal of Logic Programming* 12, 25-50.
- REED D. W. AND LOVELAND D. W. [1992b], Near-Horn Prolog and the ancestry family of procedures, Technical Report Technical report DUKE-TR-1992-20, Department of Computer Science, Duke University.
- REEVES S. [1987], Semantic tableaux as a framework for automated theorem-proving, in C. S. Mellish and J. Hallam, eds, 'Advances in Artificial Intelligence (Proceedings of AISB-87)', Wiley, pp. 125-139.
- REITER R. [1971], 'Two results on ordering for resolution with merging and linear format', *Journal of the ACM* 18, 630-646.
- ROBINSON J. A. [1965a], 'Automatic deduction with hyper-resolution', *Int. Journal of Computer Math.* 1, 227-234. Reprinted in [Siekmann and Wrightson 1983b].
- ROBINSON J. A. [1965b], 'A machine-oriented logic based on the resolution principle', *JACM* 12(1), 23-41. Reprinted in [Siekmann and Wrightson 1983b].
- SCHÜTTE K. [1960], *Beweistheorie*, Vol. 103 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen mit besonderer Berücksichtigung der Anwendungsgebiete*, Springer-Verlag.
- SHEERAN M. AND STÅLMARCK G. [2000], 'A tutorial on Stålmarck's proof procedure for propositional logic', *Formal Methods in Systems Design* 16(1), 23-58.
- SHIRAI Y. AND HASEGAWA R. [1995], Two approaches for finite-domain constraint satisfaction problem: CP and MGTP, in L. Stirling, ed., 'Proc. 12th International Conference on Logic Programming', MIT Press, pp. 249-263.
- SHULTS B. [1997], A framework for using knowledge in tableau proofs, in D. Galmiche, ed., 'Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France', Vol. 1227 of *LNCS*, Springer-Verlag, pp. 328-342.
- SIEKMANN, J. AND WRIGHTSON, G., EDS [1983a], *Automation of Reasoning: Classical Papers in Computational Logic 1967-1970*, Vol. 2, Springer-Verlag.
- SIEKMANN, J. AND WRIGHTSON, G., EDS [1983b], *Automation of Reasoning: Classical Papers in Computational Logic 1957-1966*, Vol. 1, Springer-Verlag.
- SLAGLE J. R. [1967], 'Automatic theorem proving with renamable and semantic resolution', *Journal of the ACM* 14(4), 687-697. Reprinted in [Siekmann and Wrightson 1983a].
- SMULLYAN R. M. [1963], 'A unifying principle in quantification theory', *Proceedings of the National Academy of Sciences of the U.S.A.* 49(6), 828-832.
- SMULLYAN R. M. [1995], *First-Order Logic*, second corrected edn, Dover Publications, New York. First published 1968 by Springer-Verlag.
- STICKEL M. E. [1992], 'A Prolog technology theorem prover: A new exposition and implementation in Prolog', *Theoretical Computer Science* 104(1), 109-129.
- SUTCLIFFE G. AND SUTTNER C. [1999], 'The CADE-15 ATP system competition', *Journal of Automated Reasoning* 23(1), 1-23.

- SUTTNER C. B. AND SUTCLIFFE G. [1999], The TPTP problem library, TPTP v2.2.0, Technical Report JCU-CS-99/02, Department of Computer Science, James Cook University.
- SZABO, M. E., ED. [1969], *The Collected Papers of Gerhard Gentzen*, North-Holland, Amsterdam.
- VORONKOV A. [1996], Proof search in intuitionistic logic based on constraint satisfaction, in P. Miglioli, U. Moscato, D. Mundici and M. Ornaghi, eds, 'Proc. 5th International Workshop on Theorem Proving with analytic Tableaux and Related Methods (TABLEAUX), Terrassini, Italy', Vol. 1071 of *LNCS*, Springer-Verlag, pp. 312–329.
- VORONKOV A. [1998], Herbrand's theorem, automated reasoning and semantic tableaux, in 'Proc. 13th IEEE Symposium on Logic in Computer Science, LICS, Indianapolis, USA', IEEE Press, Los Alamitos, pp. 252–263.
- WAALER A. [2001], Connections in nonclassical logics, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 22, pp. 1487–1578.
- WALLACE K. [1994], Proof Truncation Techniques in Model Elimination Tableaux, PhD thesis, University of Newcastle, Australia.
- WALLACE K. AND WRIGHTSON G. [1995], 'Regressive merging in model elimination tableau-based theorem provers', *Journal of the Interest Group in Pure and Applied Logics* 3(6), 921–938. Special Issue: Selected Papers from Tableaux'94.
- WEIDENBACH C. [2001], Combining superposition, sorts and splitting, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 27, pp. 1965–2013.

Notation

A

\mathcal{A}_Σ (atoms) 104

F

F_Σ (function symbols) 104

F_{sko} (Skolem function symbols) 107

L

\mathcal{L}_Σ (first-order formulas) 104

P

P_Σ (predicate symbols) 104

S

Σ (first-order signature) 104

Σ^* (first-order signature with Skolem
symbols) 107

T

\mathcal{T}_Σ (terms) 104

V

Var (variables) 104

Index

A

ancestor cancellation	159
ancestry family	159
atom	104

B

branch	111
closed	112, 126
open	112
subsumption	145

C

cancellation pruning rule	159
chain	153
admissible	153
clause	105
empty	105
Horn	105
instance	106
ground	106
new	106
positive	105
relevant	131
unit	105
CNF	<i>see</i> conjunctive normal form
complete	
calculus	117
strongly	117
completion mode	119
computation rule	117
fair	117
conjunctive normal form	105
connected extension step	130
connection	122
connection method	122
consolution	165
constraint model generation	143
cut	147, 148, 163

D

Davis-Putnam procedure	163
destructive	118
dilemma rule	164
disjunctive logic programming	157
domain	106

E

extension rule with local lemmas	149
--	-----

F

fact	105
------------	-----

folding up rule	150
formula	
complement	105
first-order	104
propositional	104
formula instantiation problem	165

G

goal normal form	156
------------------------	-----

H

Herbrand complexity	107
Hintikka set	120
hyper tableau extension	140
hypergraph	124

I

inheritance near-Horn Prolog	159
integrity constraint	143
interpretation	106

L

level cut	124
literal	104
ancestor	153
body	157
head	157
negative	104
ordering	136
positive	104
type <i>A</i>	153
type <i>B</i>	153
local lemma	149
logical consequence	107

M

magic set	124
mated	122
mating	122
matrix	122
spanned	123
MGU	<i>see</i> most general unifier
minimal proof length	165
minimally unsatisfiable	131
model	107
model elimination	153
restart	161
strict	161
model generation	143
most general unifier	105

N

negation normal form	104
NNF	<i>see</i> negation normal form

P

p-equivalent	165
p-simulation	165
path	122
principle of bivalence	163
proof confluent	119
pure clause rule	163

Q

query	105, 156
-------------	----------

R

range-restricted	141
reduction	124
reduction step	130
relevancy testing	124
resolution	
linear	155
Prolog-SLD	156
SLWV	137
unit	163
rule	105

S

satisfiability	107
saturation	134
scope	104
selection function	135
complete	140, 143
consistent	137, 143
stable	137
sentence	105
signature	104
simplified problem reduction format ..	160
size	
of formula	104
of tableau	112
Skolem term	110
Stålmarck's procedure	164
structure	
canonical	113
first-order	106
Herbrand	107
term domain	107
subformula	106
immediate	106
negative occurrence	106
positive occurrence	106
proper	106
substitution	105

grounding	105
idempotent	105
renaming	105

T

tableau	111
clause	126
closed	126
connection	129
EP	141
hyper	
positive	140
KE	162
proof	112, 126
proof procedure	117
regular	131
restart	162
strict	162
satisfaction	113
semantic semantic	140
subsumption	145
weak connection	131
with merging	151
with regressive merging	151
with unification	111
tableau calculus	111
tautology	105
term	104
ground	104
truth	106

U

unifier	105, 115
unit extension step	163
unit near-Horn Prolog	159

V

validity	107
variable	104
bound	105
free	105
rigid	114
universal	115
variable assignment	106
variant	106

W

well-order	144
------------------	-----

The Inverse Method

Anatoli Degtyarev

Andrei Voronkov

SECOND READERS: Ullrich Hustadt, Gregory Mints, Frank Pfenning, and Renate Schmidt.

Contents

1	Introduction	181
1.1	Backward and forward reasoning	181
1.2	The inverse method	181
1.3	Structure of the chapter	184
2	Preliminaries	185
3	Cooking classical logic	186
3.1	Ingredients: sequents and sequent calculi	186
3.2	Subformula property of C_{inv}	192
3.3	The calculus C_{inv}^{ϵ} and lifting	196
3.4	Negation normal forms	203
3.5	Saturation algorithms and linear representation of derivations	206
3.6	Exploiting other properties of sequent calculi	206
4	Applying the recipe to nonclassical logics	209
4.1	Intuitionistic logic	209
4.2	Modal logics	217
4.3	Other modal logics	219
5	Naming and connections with resolution	219
5.1	Naming	221
5.2	The inverse method as resolution	222
5.3	Optimized translation	227
5.4	Why hyperresolution	229
5.5	Inference rules not captured by hyperresolution	229
5.6	Further issues to naming	230
5.7	Fast food warning	232
6	Season your meal: strategies and redundancies	232
7	Path calculi	233
7.1	The tableau path calculus	234
7.2	Proof-theoretic properties of the tableau path calculus	237
7.3	Inverse path calculus	245
7.4	More redundancies: subsumption	248
8	Logics without the contraction rules	255

8.1	Tableau path calculus and bisimulation lemma	256
8.2	Inverse path calculus and decidability	258
9	Conclusion	260
9.1	History	260
9.2	Limits of the technique and future research	263
9.3	Warning	263
	Bibliography	264
	Index	270

1. Introduction

1.1. Backward and forward reasoning

There are several methods of theorem proving. The best known methods are resolution-based and tableau-based ones. Resolution-based methods convert the goal formula into clausal norm form and perform derivation on the set of clauses using a saturation algorithm. Tableau-based methods operate directly on formulas by reducing goals to subgoals and trying to solve the subgoals, either by reduction to new subgoals or by unification-based methods.

The inverse method is much less known. Like the tableau method, it operates on formulas, but the main operation is not reduction of goals to subgoals, but rather construction of goals from previously proved subgoals. Like resolution-based methods, the inverse method uses a saturation algorithm.

1.2. The inverse method

The inverse method is unusual compared to semantic tableaux and related methods. Instead of searching for a derivation in a goal-directed manner, by reducing the goal to subgoals until all subgoals reduce to axioms, it tries to prove the goal in the inverse direction: from axioms. Since in most calculi the number of axioms is infinite, proving theorems in the inverse direction requires one to use some properties of the calculi. Similar to the tableau method, the inverse method exploits special properties of *cut-free sequent calculi*, especially the *subformula property*.

To explain why sequent calculi are most suitable for proof-search, we consider an example rule \supset -elimination of the natural deduction system and its analogue in a sequent calculus. The natural deduction rule has the form

$$\frac{A \quad A \supset B}{B} \quad (\supset\text{-elimination}).$$

Suppose that we are given the formula B to prove. This rule says that we can try to find a formula A such that both A and $A \supset B$ are provable. There are no a priori restrictions imposed by this rule on the formula A that would allow us to restrict the search for a candidate A to a small (hopefully finite) number of formulas. Of course, one could substitute a (second-order) variable for A and try to instantiate this variable during the proof-search, like it is done in some higher-order systems, for example Isabelle [Paulson 1990], but for first-order logics this technique is not efficient compared to more specialized methods.

Consider now the corresponding rule for handling implication in the sequent calculus. We call *signed formulas* expressions of the form $T A$ or $F A$, where A is a formula. T and F are called *signs*. Think of $T A$ as saying that A is true, and $F A$ as saying that A is false. A *sequent* is a finite multiset of signed formulas. Think of a sequent $T A_1, \dots, T A_n, F B_1, \dots, F B_m$ as saying that all A_i 's are true and all B_j 's are false. The standard sequent calculus rule for handling implication corresponding to the above natural deduction rule is

$$\frac{\Gamma, FA \quad \Gamma, TB}{\Gamma, TA \supset B} (T \supset),$$

where Γ is any sequent. Let us for a moment forget about Γ and consider the simplified version of this rule:

$$\frac{FA \quad TB}{TA \supset B} (T \supset).$$

Essentially, this rule says that $A \supset B$ is true in a structure if and only if A is false or B is true. Tableau-based methods would apply this rule in the backward direction: to prove the goal $TA \supset B$, they reduce it to two subgoals FA and TB . The inverse method does the inverse: if FA and TB are already established, it will establish $TA \supset B$. Thus, the inverse method proves formulas in the *forward* direction.

At first sight this idea of the proof-search in the inverse direction reminds one of the British Museum algorithm, that generates, starting from (all possible) axioms, all provable formulas by applying all possible inference rules to already proved formulas, until the goal formula is found. Although the British Museum algorithm may in theory be better than known automated deduction methods, see [Orevkov 1983], one can hardly expect it to solve difficult problems, because of its blind, non goal-oriented behavior.

However, for some calculi we can restrict ourselves to a *strict subset* of all possible axioms and of all possible derivations. The main property of (cut-free) sequent calculi that allows us to restrict forward derivations is the *subformula property*: if a formula has a refutation, then there exists a refutation of this formula consisting only of subformulas of this formula.

1.1. EXAMPLE. Consider the (classically valid) formula $((P \supset Q) \supset P) \supset P$, where P, Q are atomic formulas. The subformula property tells us that we can restrict ourselves to the subformulas of this formula. Since implication is the only logical connective in the formula, we should only consider sequent calculus rules for handling implication. One such rule ($T \supset$) is defined above, the other rule is

$$\frac{\Gamma, TA, FB}{\Gamma, FA \supset B} (F \supset).$$

Derivations in the sequent calculus search for models of sequents. This means that to establish provability of formula (1.1), they search for models of the sequent

$$F((P \supset Q) \supset P) \supset P. \quad (1.1)$$

If this sequent has no model, then the formula $((P \supset Q) \supset P) \supset P$ is true in all structures, and therefore valid. One possible derivation of this sequent is

$$\frac{\frac{\frac{TP, FQ, FP}{FP \supset Q, FP} (F \supset) \quad TP, FP}{T(P \supset Q) \supset P, FP} (T \supset)}{F((P \supset Q) \supset P) \supset P} (F \supset).$$

It is quite clear how to obtain this derivation in a backward manner: starting with goal sequent (1.1), we can try to apply the inference rules ($T \supset$) and ($F \supset$) from their conclusions to their premises, until we obtain axioms.

The proof-search in the forward direction is less straightforward, but also possible. Using the subformula property of sequent calculi, we know that the only axioms that can be used in a derivation of (1.1) are sequents of the form Γ, TP, FP and Γ, TQ, FQ . As we will see later, TQ is *not* a signed subformula of (1.1), therefore Γ, TP, FP is the only choice. Since at this moment we have no idea about what Γ should be, we leave it out and start with the sequent

$$TP, FP. \quad (1.2)$$

Now, using this sequent, we can try to "build" larger signed subformulas of (1.1), using the rules of the sequent calculus but applied in the forward manner.

One appropriate signed subformula of (1.1) is $FP \supset Q$ that can be obtained by the following inference:

$$\frac{\Gamma, TP, FQ}{\Gamma, FP \supset Q} (F \supset).$$

However, the only sequent proved so far is (1.2) does not contain FQ . We can add FQ to (1.2) and apply this inference obtaining the sequent

$$FP, FP \supset Q. \quad (1.3)$$

This suggests that the rule $F \supset$ is not well-suited for proof-search in the forward direction and can be replaced by two more convenient rules

$$\frac{\Gamma, TA}{\Gamma, FA \supset B} (F \supset_l) \quad \text{and} \quad \frac{\Gamma, FB}{\Gamma, FA \supset B} (F \supset_r).$$

Now we can say that (1.3) is obtained from (1.2) by the rule ($F \supset_l$).

The next signed subformula of (1.1) is $T(P \supset Q) \supset P$. We can obtain this formula using the ($T \supset$) rule of sequent calculi:

$$\frac{\Gamma, TP \quad \Gamma, FP \supset Q}{\Gamma, T(P \supset Q) \supset P} (T \supset).$$

We have TP in sequent (1.2) and $FP \supset Q$ in sequent (1.3). We can apply the rule ($T \supset$) with FP substituted for Γ :

$$\frac{FP, TP \quad FP, FP \supset Q}{FP, T(P \supset Q) \supset P} (T \supset).$$

However, it is not typical for proof-search in the forward direction that we apply the rule ($T \supset$) immediately. It is more typical that we prove sequents Γ, FA and Δ, TB for *different* Γ and Δ . What we can do in such a situation is to modify the rule ($T \supset$) to be

$$\frac{\Gamma, FA \quad \Delta, TB}{\Gamma, \Delta, TA \supset B} (T \supset).$$

When we apply the modified rule to (1.2) and (1.3) we obtain

$$FP, FP, T(P \supset Q) \supset P. \quad (1.4)$$

We can continue in the same manner until we obtain a derivation of the goal sequent.

This example deals with a propositional formula and is very simple. We will show later that the idea of forward proof-search in sequent calculi can be applied to the full predicate calculus using the standard three-stage *Universal Recipe of Automated Deduction*:

- design a suitable calculus dealing with closed formulas (usually called the ground version);
- add unification and cook the calculus into a free-variable calculus;
- season it with various proof-search strategies that allow one to restrict the search space.

We will show how the Universal Recipe can be applied to various logics to obtain inverse proof procedures for these logics. Given a signed formula ξ to be proved, we define a calculus C_{inv}^{ξ} intended to search for a proof of ξ in the forward direction. In the sequel we always denote by ξ a closed signed formula and sometimes call it the *goal signed formula*, or simply the *goal*.

CONVENTION	In the sequel ξ denotes the goal signed formula.
------------	--

The main property we want to achieve is what we call the *finite rule property*. That is

- C_{inv}^{ξ} has a finite number of axioms;
- Given a finite number of sequents, there is only a finite number of inferences (i.e. one-step derivations) of C_{inv}^{ξ} applicable to these sequents.

Having such a calculus, a naive proof-search algorithm can, starting with the axioms of C_{inv}^{ξ} , repeatedly apply the inference rules of the calculus to already proved sequents, until a proof of ξ is obtained or no new sequents can be derived.

1.3. Structure of the chapter

The material of this chapter is presented as a small cookbook with recipes. It is organized as follows. In Section 2 we define the main technical notions used in this chapter: those related to multisets and substitutions. In Section 3 we describe how to cook classical logic, based on the Universal Recipe. We define the ingredients for cooking: sequents and sequent calculi, the subformula property necessary for cooking, lifting, and saturation algorithms. In Section 4 we apply the Universal

Recipe to nonclassical logics: intuitionistic logic and several modal logics. In Section 5 we discuss the technique of naming and the relation of the inverse method to resolution. To season the calculi, in Section 6 we discuss the notion of redundancy in saturation-based theorem proving. In order to formalize redundancies, in Section 7 we give a new presentation of sequent calculi using the notion of paths in formulas. The use of path calculi simplifies proofs considerably, so we obtain more tasty calculi. We give proofs of completeness for several redundancies for logic **K**. In Section 8 we discuss logics without the contraction rule and show that the inverse method gives a decision procedure for first-order versions of such logics. Finally, in Section 9 we discuss the history of the inverse method, the limits of the technique, and possible future research in the area.

2. Preliminaries

Multisets. For technical reasons, we will extensively use *finite multisets*. All precise definitions concerning multisets can be found in [Nieuwenhuis and Rubio 2001, 381] (Chapter 7 of this Handbook). If Γ is a finite multiset and γ is an element, we denote by Γ, γ or by γ, Γ the multiset obtained by adding γ to Γ . Therefore, if Γ has k copies of γ , then Γ, γ has $k + 1$ copies of γ . Likewise, the union of two multisets Γ and Δ is denoted by Γ, Δ . If Γ has k copies of an element γ and Δ has l copies of γ , then Γ, Δ has $k + l$ copies of γ . Finally, we write $\Gamma \dot{\subseteq} \Delta$ to denote that Γ is a *submultiset* of Δ , i.e., for every element γ , if γ occurs in Γ k times, then γ occurs in Δ at least k times.

The *multiset difference* of multisets Γ and Δ is denoted $\Gamma \dot{-} \Delta$. If an element occurs in Γ k times and in Δ l times and $k > l$, then it occurs in $\Gamma \dot{-} \Delta$ $k - l$ times, otherwise it does not occur in $\Gamma \dot{-} \Delta$ at all. If A is a member of a multiset Γ , we will write $A \dot{\in} \Gamma$.

Expression. By an *expression* we mean a term, a formula, a tuple of terms or formulas, a multiset of terms or formulas and so on. Given an expression E we denote by $\text{var}(E)$ the set of all variables occurring in E and by $\text{free}(E)$ the set of all *free* variables of E . If $\text{var}(E) = \emptyset$ then E is called *ground*, if $\text{free}(E) = \emptyset$ then E is called *closed*.

Substitutions. We consider a *substitution* as a finite mapping from variables to terms and denote it by $\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$. It is assumed that the variables x_1, \dots, x_n are different and $x_i \neq t_i$ for all $i = 1, \dots, n$. Given a substitution θ we denote by $\text{dom}(\theta)$ its *domain*, i.e. the set of variables $\{x \mid \theta(x) \neq x\}$, and by $\text{ran}(\theta)$ its *range*, i.e. the set of terms $\{\theta(x) \mid x \in \text{dom}(\theta)\}$. Thus $\text{dom}(\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}) = \{x_1, \dots, x_n\}$, $\text{ran}(\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}) = \{t_1, \dots, t_n\}$. By $\text{vran}(\theta)$ we denote the *variable range* of θ , i.e. the set of variables which appear in a term from $\text{ran}(\theta)$, $\text{vran}(\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}) = \text{var}(t_1, \dots, t_n)$. A substitution σ is *grounding* for a set of variables V if for every variable $v \in V$ the term $v\sigma$ is ground.

A substitution is grounding for an expression E if it is grounding for $\text{free}(E)$. The **empty substitution**, denoted ε , is the only substitution with $\text{dom}(\varepsilon) = \emptyset$.

A substitution $\theta = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ is **admissible** for an expression E , if E has no occurrences of a formula $\forall y A$ or $\exists y A$ such that for some i we have $x_i \in \text{free}(\forall y A)$ or $x_i \in \text{free}(\exists y A)$, and $y \in \text{var}(t_i)$. In other words, for every $i \in \{1, \dots, n\}$ the **term t_i is free for x_i in E** , see [Kleene 1967, page 94].

For an expression E and a substitution θ admissible for E , $E\theta$ stands for the result of applying θ to E . It is obtained from E by the simultaneous replacement of free occurrences of x_1, \dots, x_n by t_1, \dots, t_n , respectively.

Let E be an expression. Following [Kleene 1967], whenever we introduce a notation like $E(x_1, \dots, x_n)$, $n \geq 1$, showing variables x_1, \dots, x_n in the notation, we will thereafter understand by $E(t_1, \dots, t_n)$ the result of application of the substitution $\theta = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ to E under the condition that θ is admissible for E . Of course, it is not required that the expression denoted by $E(x_1, \dots, x_n)$ actually contains x_i free, also it is not excluded that $E(x_1, \dots, x_n)$ may contain free variables other than x_1, \dots, x_n .

The notation \doteq stands for “equal by definition”.

The **composition of substitutions** σ and θ , denoted $\sigma\theta$, is the substitution defined by $x(\sigma\theta) \doteq (x\sigma)\theta$, for every variable x .

Let σ_1, σ_2 be substitutions such that $\text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset$. By $\sigma_1 \cup \sigma_2$ we denote the substitution σ such that $\text{dom}(\sigma) \doteq \text{dom}(\sigma_1) \cup \text{dom}(\sigma_2)$ and

$$x\sigma \doteq \begin{cases} x\sigma_1, & \text{if } x \in \text{dom}(\sigma_1), \\ x\sigma_2, & \text{if } x \in \text{dom}(\sigma_2), \end{cases}$$

for all $x \in \text{dom}(\sigma)$.

We will need more definitions concerning substitutions and unification. They will be introduced later, as soon as they are needed.

3. Cooking classical logic

In this section we apply the universal recipe of automated deduction to cook the inverse method for classical logic.

3.1. Ingredients: sequents and sequent calculi

3.1. DEFINITION (truth, satisfiability, validity). A closed signed formula $\text{T } A$ (respectively, $\text{F } A$) is **true** in a structure if so is the formula A (respectively, $\neg A$). A closed sequent is **true** in a structure if so is every signed formula in this sequent. If a closed signed formula (respectively, a closed sequent) is true in a structure, we say that the structure is a **model** of the signed formula (respectively, the sequent). A sequent is **satisfiable** if it has a model.

$$\begin{array}{c}
\frac{}{\Gamma A, FA} (Ax) \\
\frac{\Gamma, \gamma, \gamma}{\Gamma, \gamma} (C) \\
\frac{\Gamma, TA}{\Gamma, TA \wedge B} (T \wedge_l) \qquad \frac{\Gamma, TB}{\Gamma, TA \wedge B} (T \wedge_r) \\
\frac{\Gamma, FA \quad \Delta, FB}{\Gamma, \Delta, FA \wedge B} (F \wedge) \\
\frac{\Gamma, TA \quad \Delta, TB}{\Gamma, \Delta, TA \vee B} (T \vee) \\
\frac{\Gamma, FA}{\Gamma, FA \vee B} (F \vee_l) \qquad \frac{\Gamma, FB}{\Gamma, FA \vee B} (F \vee_r) \\
\frac{\Gamma, FA \quad \Delta, TB}{\Gamma, \Delta, TA \supset B} (T \supset) \\
\frac{\Gamma, TA}{\Gamma, FA \supset B} (F \supset_l) \qquad \frac{\Gamma, FB}{\Gamma, FA \supset B} (F \supset_r) \\
\frac{\Gamma, FA}{\Gamma, T \neg A} (T \neg) \qquad \frac{\Gamma, TA}{\Gamma, F \neg A} (F \neg) \\
\frac{\Gamma, TA(t)}{\Gamma, T \forall x A(x)} (T \forall) \qquad \frac{\Gamma, F A(y)}{\Gamma, F \forall x A(x)} (F \forall) \\
\frac{\Gamma, TA(y)}{\Gamma, T \exists x A(x)} (T \exists) \qquad \frac{\Gamma, F A(t)}{\Gamma, F \exists x A(x)} (F \exists)
\end{array}$$

In the rule (Ax) , A is an atomic formula. In the rules $(F\forall)$ and $(T\exists)$ the variable y is free for x in $A(x)$ and $A(y) = A(x)\{x \mapsto y\}$, furthermore y has no free occurrences in the conclusions of the rules. The variable y is called the *eigenvariable* of these rules. This condition on the rules $(F\forall)$ and $(T\exists)$ is called the *eigenvariable condition*. The rule (C) is called *contraction*.

Figure 1: Calculus \mathbf{C}_{inv}

All sequent calculi introduced in this chapter establish the unsatisfiability of sequents. In view of this fact, proofs in these calculi will sometimes be called *refutations*.

We will denote sequents by Γ, Δ, ∇ . A detailed discussion of sequent calculi can be found in [Degtyarev and Voronkov 2001a] (Chapter 10 of this Handbook). The *sequent calculus* \mathbf{C}_{inv} for classical logic is shown in Figure 1.

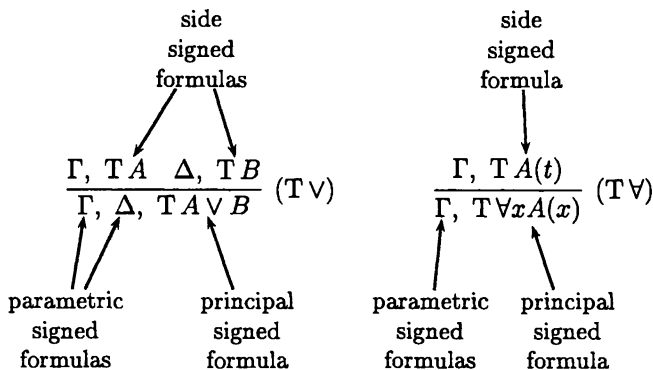


Figure 2: Principal, side and parametric signed formulas

3.2. NOTE. The calculus C_{inv} is one of the calculi developed from the original calculus LK of Gentzen [1934]. In formalizations of sequent calculi developed for semantic tableaux, LK is usually defined using *invertible rules*, following calculi introduced in [Kanger 1957, Kanger 1963], see [Degtyarev and Voronkov 2001a] (Chapter 10 of this Handbook). The discussion in the introduction on pages 182–184 reveals why we formulate the rules of C in a noninvertible way.

Let us introduce some standard definitions related to rules of the sequent calculi illustrated by Figure 2, adapted from [Kleene 1967]. Every rule infers a sequent from one or two sequents.

3.3. DEFINITION (*principal, side, and parametric formulas*). The new signed formula introduced by the rule is called the **principal signed formula** of the rule. The signed formula or formulas that occur in the premises of the rule and do not occur in the conclusion are called the **side signed formulas** of this rule. All other signed formulas, denoted by Γ, Δ , are called **parametric signed formulas** of the rule.

Since we will establish a lot of proof-theoretic results about derivations in various sequent calculi, we introduce terminology related to derivations. A similar terminology is used in [Bachmair and Ganzinger 2001] (Chapter 2 of this Handbook).

3.4. DEFINITION (*inference, derivation, refutation*). An **inference** in C_{inv} is any instance of an inference rule. A **derivation of a sequent** Γ is any tree of sequents formed by inferences and having Γ as the root. A **refutation of** Γ is any derivation of Γ whose leaves are axioms.

We will use the notions of inference, derivation and refutation for all other calculi introduced in this chapter. When we give examples of derivations, we will put side formulas of all inferences in shaded boxes. We will also omit the inferences by (Ax) .

3.5. EXAMPLE. Consider the refutation of formula (1.1) in C_{inv} :

$$\begin{array}{c}
 \frac{TP, FP}{FP \supset Q, FP} (F \supset_l) \quad \frac{TP, FP}{T(P \supset Q) \supset P, FP, FP} (T \supset) \\
 \frac{T(P \supset Q) \supset P, FP}{T(P \supset Q) \supset P, FP} (C) \\
 \frac{T(P \supset Q) \supset P, FP}{F((P \supset Q) \supset P) \supset P, FP} (F \supset_l) \\
 \frac{F((P \supset Q) \supset P) \supset P, F((P \supset Q) \supset P) \supset P}{F((P \supset Q) \supset P) \supset P} (F \supset_r) \\
 \frac{F((P \supset Q) \supset P) \supset P}{F((P \supset Q) \supset P) \supset P} (C)
 \end{array}$$

Suppose we have a calculus C for some logic L . Logics are usually defined using a suitable semantics, in this chapter we assume that the semantics defines the notion of *satisfiable sequent*. A calculus C is called *sound* for L if every sequent that has a refutation in C is unsatisfiable in L . A calculus C is called *complete for L* if every unsatisfiable sequent of L has a refutation in C . Sometimes, we will deal with weaker notions of completeness, when only unsatisfiable sequents consisting of one formula are guaranteed to have a refutation. Usually, inference rules in a sound calculus have the following property (which guarantees soundness): if all premises of this rule are unsatisfiable, then so is the conclusion. Since soundness will be more or less obvious for most calculi of this chapter, we will mainly speak about completeness, even when we mean soundness and completeness.

An inference rule is called *invertible* if it has the converse property: if the conclusion of the rule is unsatisfiable, then so are all premises of the rule. Many standard completeness rules for tableau calculi are based on calculi in which all or nearly all inference rules are invertible.

Since the rules of C_{inv} are not invertible, the standard completeness proofs are not easily applicable to it. Moreover, C_{inv} is *incomplete* for refuting sequents, but complete for refuting formulas. Let us show the technique that allows to establish completeness by purely proof-theoretic arguments. The technique is based on the following observation.

Take any cut-free sequent calculus for classical logic that is known to be complete. As an example, we take the calculus C_{tab} with invertible rules used in some formalizations of the tableau method. The calculus C_{tab} is shown in Figure 3. This calculus is essentially the calculus of Kanger [1957] or $G4$ of Kleene [1967]. The invertibility of rules of C_{tab} allows us to prove completeness using the standard arguments based on the Hintikka sets [Hintikka 1955], see also [Kanger 1957, Kleene 1967, Smullyan 1968, Fitting 1996].

3.6. THEOREM (Completeness of C_{tab}). *Let Γ be a closed sequent. Γ is unsatisfiable if and only if there exists a refutation of Γ in C_{tab} .* \square

The rules of C_{tab} , except for the rules for negation and two of the quantifier rules, are different from those of C_{inv} . Let us think how we can use completeness of C_{tab}

$$\begin{array}{c}
\frac{}{\Gamma, TA, FA} (Ax) \\
\\
\frac{\Gamma, TA, TB}{\Gamma, TA \wedge B} (T \wedge) \qquad \frac{\Gamma, FA \quad \Gamma, FB}{\Gamma, FA \wedge B} (F \wedge) \\
\\
\frac{\Gamma, TA \quad \Gamma, TB}{\Gamma, TA \vee B} (T \vee) \qquad \frac{\Gamma, FA, FB}{\Gamma, FA \vee B} (F \vee) \\
\\
\frac{\Gamma, FA \quad \Gamma, TB}{\Gamma, TA \supset B} (T \supset) \qquad \frac{\Gamma, TA, FB}{\Gamma, FA \supset B} (F \supset) \\
\\
\frac{\Gamma, FA}{\Gamma, T \neg A} (T \neg) \qquad \frac{\Gamma, TA}{\Gamma, F \neg A} (F \neg) \\
\\
\frac{\Gamma, T \forall x A(x), TA(t)}{\Gamma, T \forall x A(x)} (T \forall) \qquad \frac{\Gamma, FA(y)}{\Gamma, F \forall x A(x)} (F \forall) \\
\\
\frac{\Gamma, TA(y)}{\Gamma, T \exists x A(x)} (T \exists) \qquad \frac{\Gamma, F \exists x A(x), FA(t)}{\Gamma, F \exists x A(x)} (F \exists)
\end{array}$$

In the rule (Ax) , A is an atomic formula. The rules $(F \forall)$ and $(T \exists)$ satisfy the eigenvariable condition.

Figure 3: Calculus \mathbf{C}_{tab}

to prove completeness of \mathbf{C}_{inv} . We have a problem already with axioms. Axioms of \mathbf{C}_{tab} have the form

$$\frac{}{\Gamma, TA, FA} (Ax).$$

As a matter of fact, *not every* sequent Γ, TA, FA has a refutation in \mathbf{C}_{inv} . A simplest example is TP, FP, TQ , where P, Q are distinct 0-ary relation symbols. (Note that this shows that not every unsatisfiable sequent has a refutation in \mathbf{C}_{inv} .) However, we can derive all sequents of the form TA, FA using the axioms of \mathbf{C}_{inv} . This means that for every axiom Δ of \mathbf{C}_{tab} there exists $\Delta' \subseteq \Delta$ such that Δ' has a refutation in \mathbf{C}_{inv} . We generalize this observation to the following fact that will imply completeness of \mathbf{C}_{inv} for formulas in a rather straightforward way.

3.7. LEMMA (Subsequent Lemma). *For every sequent Γ that has a refutation in \mathbf{C}_{tab} there exists a sequent Δ such that $\Delta \subseteq \Gamma$ and Δ has refutation in \mathbf{C}_{inv} .*

PROOF. The proof is by induction on the length of a refutation of Γ in \mathbf{C}_{tab} . We consider several cases, the other cases will be similar.

CASE (Ax) . This case is obvious.

CASE $(F \supset)$. The rule of \mathbf{C}_{tab} has the form

$$\frac{\Gamma, TA, FB}{\Gamma, FA \supset B} (F \supset).$$

By the induction hypothesis, there exists a sequent $\Delta \subseteq (\Gamma, TA, FB)$ which has a refutation in \mathbf{C}_{inv} . We have to find a sequent $\Delta' \subseteq (\Gamma, FA \supset B)$ which has a refutation in \mathbf{C}_{inv} . Consider the following cases, depending on whether TA and FB occur in Δ .

SUBCASE $((TA, FB) \subseteq \Delta)$. We have $\Delta = (\nabla, TA, FB)$, for some $\nabla \subseteq \Gamma$. The following derivation in \mathbf{C}_{inv} proves this case:

$$\frac{\frac{\frac{\nabla, TA, FB}{\nabla, FA \supset B, FB} (F \supset_l)}{\nabla, FA \supset B, FA \supset B} (F \supset_r)}{\nabla, FA \supset B} (C).$$

This case also demonstrates the need for the contraction rule in \mathbf{C}_{inv} .

SUBCASE $(TA \in \Delta \text{ and } FB \notin \Delta)$. We have $\Delta = (\nabla, TA)$, for some $\nabla \subseteq \Gamma$. The following derivation in \mathbf{C}_{inv} proves this case:

$$\frac{\nabla, TA}{\nabla, FA \supset B} (F \supset_l).$$

SUBCASE $(FB \in \Delta \text{ and } TA \notin \Delta)$. This case is similar to the previous one.

SUBCASE $(TA \notin \Delta \text{ and } FB \notin \Delta)$. In this case we let $\Delta' = \Delta$.

CASE $(T \supset)$. The rule of \mathbf{C}_{tab} has the form

$$\frac{\Gamma, FA \quad \Gamma, TB}{\Gamma, TA \supset B} (T \supset).$$

By the induction hypothesis, there exist sequents $\Delta_1 \subseteq (\Gamma, FA)$ and $\Delta_2 \subseteq (\Gamma, TB)$ that have refutations in \mathbf{C}_{inv} . We have to find a sequent $\Delta \subseteq (\Gamma, TA \supset B)$ refutable in \mathbf{C}_{inv} .

If we have $\Delta_1 \subseteq \Gamma$, then we can take Δ_1 to be Δ . Likewise, if we have $\Delta_2 \subseteq \Gamma$, then we can take Δ_2 to be Δ . Therefore, we can assume that $\Delta_1 = (\Delta'_1, FA)$ and $\Delta_2 = (\Delta'_2, TB)$ for some $\Delta'_1 \subseteq \Gamma$ and $\Delta'_2 \subseteq \Gamma$. Since $\Delta'_1 \subseteq \Gamma$ and $\Delta'_2 \subseteq \Gamma$, some sequent $\Gamma' \subseteq \Gamma$ can be obtained from the sequent Δ'_1, Δ'_2 in \mathbf{C}_{inv} by a series of contractions. The following derivation in \mathbf{C}_{inv} proves this case:

$$\frac{\Delta'_1, FA \quad \Delta'_2, TB}{\Delta'_1, \Delta'_2, TA \supset B} (T \supset)$$

$$\vdots \text{ series of contractions}$$

$$\Gamma', TA \supset B$$

□

The name *subsequent lemma* is due to the fact that we prove the existence of a subsequent Δ of Γ .

Using this lemma, we obtain completeness of \mathbf{C}_{inv} for formulas:

3.8. THEOREM (Completeness of \mathbf{C}_{inv}). *Let ξ be a closed signed formula. ξ is unsatisfiable if and only if it has a refutation in \mathbf{C}_{inv} .*

PROOF. By the Completeness Theorem 3.6 for \mathbf{C}_{tab} , ξ is unsatisfiable if and only if there exists a refutation of ξ in \mathbf{C}_{tab} . By Subsequent Lemma 3.7, there exists such a refutation of ξ if and only if some submultiset of ξ has a refutation in \mathbf{C}_{inv} . Since the empty sequent has no refutation in \mathbf{C}_{inv} , the only submultiset of ξ refutable in \mathbf{C}_{inv} is ξ itself. Therefore, ξ is unsatisfiable if and only if it has a refutation in \mathbf{C}_{inv} . \square

3.9. EXAMPLE. Consider the signed formula

$$\xi = F \forall y \exists x \exists z (P(z, x) \supset P(z, fx) \wedge P(x, ffy)). \quad (3.1)$$

(We omit parentheses in terms like $f(f(y))$ in this and other examples.) A refutation of ξ in \mathbf{C}_{tab} is shown in Figure 4. Applying the algorithm implicit in the proof of the Subsequent Lemma to this derivation, we obtain the derivation of (3.1) in \mathbf{C}_{inv} shown in Figure 5.

3.2. Subformula property of \mathbf{C}_{inv}

Following the universal recipe of automated deduction, we have obtained a suitable ground version \mathbf{C}_{inv} of a sequent calculus intended to be used for a forward proof-search. The next step is to obtain a free-variable version of \mathbf{C}_{inv} . In the free-variable version, automatic refutations of a goal formula G will be obtained by repeatedly applying inference rules of the free-variable calculus in the forward direction to initial sequents (axioms). Evidently, it is desirable to restrict ourselves to a finite number of axioms. To this end, we will use the subformula property of \mathbf{C}_{inv} : a derivation of a signed formula ξ may only involve signed subformulas of ξ .

There are two standard definitions of subformulas. The first one defines subformulas of $\exists x A(x)$ as $A(x)$ plus subformulas of $A(x)$. The second one used e.g. in [Gentzen 1934], defines subformulas of $\exists x A(x)$ as all formulas of the form $A(t)$, where t is a term, plus subformulas of $A(t)$. We use both definitions, but to distinguish between them we call subformulas of the first kind *free subformulas*. In order to give the right definition of a signed subformula, it is enough to analyze the rules of \mathbf{C}_{inv} (or, alternatively, the rules of \mathbf{C}_{tab}).

3.10. DEFINITION (signed subformula). The notion of a *free signed subformula* of a signed formula α is defined by induction as follows. First, we define *free immediate signed subformulas* and *immediate signed subformulas* using the table of Figure 6.

In the table below, t is an arbitrary term free for x in $A(x)$.

signed formula	its free immediate signed subformulas	its immediate signed subformulas
$TA \wedge B$	TA and TB	TA and TB
$FA \wedge B$	FA and FB	FA and FB
$TA \vee B$	TA and TB	TA and TB
$FA \vee B$	FA and FB	FA and FB
$TA \supset B$	FA and TB	FA and TB
$FA \supset B$	TA and FB	TA and FB
$T\neg A$	FA	FA
$F\neg A$	TA	TA
$T\exists xA(x)$	$TA(x)$	$TA(t)$
$F\exists xA(x)$	$FA(x)$	$FA(t)$
$T\forall xA(x)$	$TA(x)$	$TA(t)$
$F\forall xA(x)$	$FA(x)$	$FA(t)$

Figure 6: Immediate signed subformulas

For all signed formulas α, β, γ we define various notions of subformulas as follows.

1. α is a (free) signed subformula of α ;
2. If α is a (free) immediate signed subformula of β and β is a (free) signed subformula of γ , then α is a (free) signed subformula of γ .

We already mentioned the **subformula property** several times without defining it. There can be several definitions of the subformula property. Below we give several forms of subformula property, the first is the strongest.

- *Every derivation* of a sequent Γ consists of signed subformulas of signed formulas in Γ ;
- *Every refutation* of a sequent Γ consists of signed subformulas of signed formulas in Γ ;
- If Γ has a refutation, then *there exists* a refutation of Γ consisting of signed subformulas of signed formulas in Γ .
- *Every derivation* of a sequent Γ consists of subformulas of signed formulas in Γ .

All free-variable calculi discussed in this chapter have the fourth form of the subformula property. All other calculi of this chapter have the first, strongest form.

3.11. THEOREM (Subformula Property of \mathbf{C}_{inv}). *Let Π be a derivation of a signed formula ξ in \mathbf{C}_{inv} . Every signed formula occurring in Π is a signed subformula of ξ .*

PROOF. By a routine inspection of inference rules of C_{inv} . \square

Thus, when we search for a refutation of a particular signed formula ξ , we can restrict our search to sequents consisting of signed subformulas of ξ . When ξ contains quantifiers, it may have an infinite number of signed subformulas, so the subformula property does not restrict the search space good enough. The next thing to note is that signed subformulas of ξ have a convenient representation in terms of *free* signed subformulas, and any signed formula has only a finite number of free signed subformulas.

3.2.1. Representation of subformulas by free subformulas

We call a (signed) formula α **rectified** if (i) every occurrence of a quantifier in α binds a different variable and (ii) every bound variable of α has no free occurrence in α . Evidently, we can make any (signed) formula into an equivalent rectified one by renaming of bound variables. A sequent Γ is **rectified** if so is every signed formula in Γ .

CONVENTION

In the sequel we assume all formulas to be rectified.

We will often represent signed subformulas of a given formula ξ in the form $\alpha \cdot \theta$, where α is a free signed subformula of ξ and θ is a substitution admissible for A . We say that a pair $\alpha \cdot \theta$ **represents** the signed subformula $\alpha\theta$. We also call this representation the representation **via a free signed subformula**. Note that the same signed subformula β can have different representations via different free signed subformulas.

We call a substitution θ **relevant** to an expression E if $\text{dom}(\theta) \subseteq \text{free}(E)$. We introduce a convention of a technical nature. Whenever we use the notation $\alpha \cdot \theta$ such that θ is not relevant to α , we mean the representation $\alpha \cdot \tau$, where τ is the restriction of θ on $\text{free}(\alpha)$. This convention will considerably simplify notation.

3.12. LEMMA. *Let α, β be rectified signed formulas. Then β is a signed subformula of α if and only if $\beta = \gamma\theta$ for some free signed subformula γ of α and substitution θ such that $\text{dom}(\theta) \cap \text{free}(\alpha) = \emptyset$. For given β and γ such a substitution θ is unique among substitutions relevant to γ .*

PROOF. By routine inspection of the definition of signed subformulas. \square

In particular, every signed subformula of a closed signed formula can be obtained from a free signed subformula by applying a substitution. Now we can reformulate the subformula property of C_{inv} in the following way.

3.13. COROLLARY. *Let Π be a refutation of a rectified closed signed formula ξ in C_{inv} . Every signed formula occurring in Π has the form $\gamma\theta$ for a free signed subformula γ of ξ and a substitution θ .* \square

Suppose that we want to check the refutability in C_{inv} of a closed signed formula ξ . By Corollary 3.13, we can restrict signed formulas occurring in the derivation to signed formulas of the form $\gamma\theta$, where γ is a free signed subformula of ξ and θ is a substitution. Since this applies to axioms as well, every axiom has the form $TA\theta$, $FB\sigma$, where A, B are atomic and TA , FB are *free* signed subformulas of ξ . For any given A, B , there may be an infinite number of such axioms because of different choices for substitutions θ, σ , but there is only a *finite* number of pairs of *free* signed subformulas TA and FB of Σ . As usual in automated deduction, we can choose a *most general* axiom that represents all axioms $TA\theta$, $FB\sigma$ for given A and B .

3.3. The calculus C_{inv}^ξ and lifting

We will now introduce the main calculus C_{inv}^ξ for the inverse method for classical logic. The calculus is based on the idea of representing sequents through free subformulas and using most general unifiers instead of arbitrary substitutions.

As usual, we assume that ξ is a rectified closed signed formula (the goal). A *sequent of C_{inv}^ξ* is any multiset of pairs $\alpha \cdot \sigma$ that represent subformulas of ξ . We say that a sequent Γ of C_{inv} is an *instance* of a sequent $\alpha_1 \cdot \sigma_1, \dots, \alpha_n \cdot \sigma_n$ of C_{inv}^ξ if there exists a substitution τ such that the multiset $\alpha_1 \sigma_1 \tau, \dots, \alpha_n \sigma_n \tau$ coincides with Γ .

Now our aim is the following. Given a closed signed formula ξ , to make a sequent calculus C_{inv}^ξ that has the following property: for every sequent Γ refutable in C_{inv} and consisting of signed subformulas of ξ , there exists a sequent Δ refutable in C_{inv}^ξ such that Γ is an instance of Δ . Before defining such a calculus we have to introduce more definitions.

A *restriction* of a substitution σ to a set of variables V , denoted by $\sigma|_V$, is the substitution defined as follows. For every variable x

$$x(\sigma|_V) = \begin{cases} x\sigma, & \text{if } x \in V; \\ x, & \text{otherwise.} \end{cases}$$

We denote by σ_{-x} the substitution $\sigma|_{\text{dom}(\sigma) \setminus \{x\}}$. It is easy to see that

$$y\sigma_{-x} = \begin{cases} x, & \text{if } x = y; \\ y\sigma, & \text{otherwise.} \end{cases}$$

A substitution σ is *more general* than a substitution θ , denoted $\sigma \leq \theta$ if there exists a substitution τ such that $\sigma\tau = \theta$.

We introduce the notions related to *unification* of expressions and substitutions.

3.14. DEFINITION (unification). Let E_1, E_2 be two expressions and θ a substitution admissible for E_1, E_2 . We call θ a *unifier* of E_1 and E_2 if $E_1\theta = E_2\theta$. Two expressions are *unifiable* if they have a unifier. A unifier θ of E_1 and E_2 is called

most general, denoted $\text{mgu}(E_1, E_2)$, if θ is more general than any unifier of E_1 and E_2 .

A **most general unifier of substitutions** σ and τ , denoted $\text{mgu}(\sigma, \tau)$, is a substitution θ defined as follows. Let $\{x_1, \dots, x_n\} = \text{dom}(\sigma) \cup \text{dom}(\tau)$, $n \geq 0$. Then θ is any most general unifier of the tuples $\langle x_1\sigma, \dots, x_n\sigma \rangle$ and $\langle x_1\tau, \dots, x_n\tau \rangle$.

A substitution ρ is called a **renaming** if ρ is a one-to-one mapping from its domain to itself. It is easy to see that a substitution ρ is a renaming if and only if $\text{dom}(\rho) = \text{ran}(\rho)$. Also ρ is a renaming if and only if there exists a substitution τ such that $\rho\tau = \varepsilon$ (or $\tau\rho = \varepsilon$). Let us note, that if a substitution σ is a most general unifier of terms t and s , then $\sigma\rho$ is also a most general unifier of these terms for any renaming ρ .

3.15. DEFINITION (free variables, variants, renaming). Let $\Gamma = \alpha_1 \cdot \sigma_1, \dots, \alpha_n \cdot \sigma_n$. The set of **free variables** of Γ , denoted by $\text{free}(\Gamma)$, is the set of all free variables of $\alpha_1\sigma_1, \dots, \alpha_n\sigma_n$. For a substitution θ we denote by $\Gamma\theta$ the sequent $\alpha_1 \cdot \sigma_1\theta, \dots, \alpha_n \cdot \sigma_n\theta$. We call a **variant** of Γ any sequent $\Gamma\rho$, where ρ is a renaming. Two expressions E_1 and E_2 are **variable-disjoint** if $\text{free}(E_1) \cap \text{free}(E_2) = \emptyset$. We say that a substitution ρ **renames E_1 away from E_2** if ρ is a renaming such that $E_1\rho$ and E_2 are variable-disjoint. If, for such a renaming, $E_1\rho$ and E_2 are unifiable, we say that E_1 and E_2 are **weakly unifiable**.

Like in standard resolution calculi, we identify sequents that are variants of each other. Therefore, we assume an implicit **renaming rule**:

$$\frac{\Gamma}{\Gamma\rho}, \text{ where } \rho \text{ is a renaming.}$$

The calculus \mathbf{C}_{inv}^ξ is given in Figure 7. An example derivation in \mathbf{C}_{inv}^ξ , when ξ is (3.1), is shown in Figure 8. Compare this derivation with that of Figure 5 on page 193.

We can prove completeness of \mathbf{C}_{inv}^ξ by comparing \mathbf{C}_{inv} with \mathbf{C}_{inv}^ξ and using lifting arguments, expressed by the following lemma.

3.16. LEMMA (Instance Lemma for \mathbf{C}_{inv}^ξ). *Let ξ be a closed signed formula and let $\alpha_1, \dots, \alpha_n$ be free signed subformulas of ξ . Let $\theta_1, \dots, \theta_n$ be substitutions such that the sequent $\alpha_1\theta_1, \dots, \alpha_n\theta_n$ is derivable in \mathbf{C}_{inv} . Then there exist substitutions $\lambda, \tau_1, \dots, \tau_n$ such that the sequent $\alpha_1 \cdot \tau_1, \dots, \alpha_n \cdot \tau_n$ is derivable in \mathbf{C}_{inv}^ξ and $\alpha_i\tau_i\lambda = \alpha_i\theta_i$ for all $i \in \{1, \dots, n\}$.*

PROOF. The proof is by induction on the length of a derivation Π in \mathbf{C}_{inv} . We will consider the last of inference of Π and assume, by induction hypothesis, that the lemma holds for all the premises of this inference. We consider several cases, depending on the last inference, the other cases will be similar. These cases are the following:

- the last rule of Π is (Ax) ;

All signed formulas in the inference rules are signed subformulas of ξ . In the rule (Ax) below, A and B are weakly unifiable atoms, ρ renames away A from B and τ is a most general unifier of $A\rho$ and B . In all rules the premises are variable-disjoint. In all rules each premise is also variable-disjoint with $\text{var}(\xi)$. In all rules θ is a most general unifier of σ_1, σ_2 . The rules $(F\forall)$ and $(T\exists)$ satisfy the eigenvariable condition: the term $x\sigma$ is a variable not occurring in the conclusion of the rule.

$$\begin{array}{c}
\overline{TA \cdot \rho\tau, FB \cdot \tau} \quad (Ax) \\
\\
\frac{\Gamma, \gamma \cdot \sigma_1, \gamma \cdot \sigma_2}{\Gamma\theta, \gamma \cdot \sigma_1\theta} \quad (C) \\
\\
\frac{\Gamma, TA \cdot \sigma}{\Gamma, TA \wedge B \cdot \sigma} \quad (T\wedge_l) \qquad \frac{\Gamma, TA \cdot \sigma}{\Gamma, TB \wedge A \cdot \sigma} \quad (T\wedge_r) \\
\\
\frac{\Gamma, FA \cdot \sigma_1 \quad \Delta, FB \cdot \sigma_2}{\Gamma\theta, \Delta\theta, FA \wedge B \cdot \sigma_1\theta} \quad (F\wedge) \\
\\
\frac{\Gamma, TA \cdot \sigma_1 \quad \Delta, TB \cdot \sigma_2}{\Gamma\theta, \Delta\theta, TA \vee B \cdot \sigma_1\theta} \quad (T\vee) \\
\\
\frac{\Gamma, FA \cdot \sigma}{\Gamma, FA \vee B \cdot \sigma} \quad (F\vee_l) \qquad \frac{\Gamma, FB \cdot \sigma}{\Gamma, FA \vee B \cdot \sigma} \quad (F\vee_r) \\
\\
\frac{\Gamma, FA \cdot \sigma_1 \quad \Delta, TB \cdot \sigma_2}{\Gamma\theta, \Delta\theta, TA \supset B \cdot \sigma_1\theta} \quad (T\supset) \\
\\
\frac{\Gamma, TA \cdot \sigma}{\Gamma, FA \supset B \cdot \sigma} \quad (F\supset_l) \qquad \frac{\Gamma, FB \cdot \sigma}{\Gamma, FA \supset B \cdot \sigma} \quad (F\supset_r) \\
\\
\frac{\Gamma, FA \cdot \sigma}{\Gamma, T\neg A \cdot \sigma} \quad (T\neg) \qquad \frac{\Gamma, TA \cdot \sigma}{\Gamma, F\neg A \cdot \sigma} \quad (F\neg) \\
\\
\frac{\Gamma, TA \cdot \sigma}{\Gamma, T\forall x A \cdot \sigma_{-x}} \quad (T\forall) \qquad \frac{\Gamma, FA \cdot \sigma}{\Gamma, F\forall x A \cdot \sigma_{-x}} \quad (F\forall) \\
\\
\frac{\Gamma, TA \cdot \sigma}{\Gamma, T\exists x A \cdot \sigma_{-x}} \quad (T\exists) \qquad \frac{\Gamma, FA \cdot \sigma}{\Gamma, F\exists x A \cdot \sigma_{-x}} \quad (F\exists)
\end{array}$$

Figure 7: Calculus \mathbf{C}_{inv}^ξ

$$\begin{array}{c}
\frac{TP(z, x) \cdot \{x \mapsto fx_2, z \mapsto x_1\}, \overline{FP(z, fx) \cdot \{x \mapsto x_2, z \mapsto x_1\}} \quad TP(z, x) \cdot \{x \mapsto ffy_2, z \mapsto y_1\}, \overline{FP(x, ffy) \cdot \{x \mapsto y_1, y \mapsto y_2\}}}{\overline{TP(z, x) \cdot \{x \mapsto fx_2, z \mapsto x_1\}, \overline{TP(z, x) \cdot \{x \mapsto ffx_3, z \mapsto x_2\}}, \overline{FP(z, fx) \wedge P(x, ffy) \cdot \{x \mapsto x_2, y \mapsto x_3, z \mapsto x_1\}}}} (F \wedge) \\
\frac{TP(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \overline{FP(z, fx) \wedge P(x, ffy) \cdot \{x \mapsto fx_1, y \mapsto x_1, z \mapsto fx_1\}}}{TP(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \overline{FP(z, fx) \wedge P(x, ffy) \cdot \{x \mapsto fx_1, y \mapsto x_1, z \mapsto fx_1\}}} (F \supset_r) \\
\frac{TP(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \overline{FP(z, x) \supset P(z, fx) \wedge P(x, ffy) \cdot \{x \mapsto fx_1, y \mapsto x_1, z \mapsto fx_1\}}}{TP(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \overline{FP(z, x) \supset P(z, fx) \wedge P(x, ffy) \cdot \{x \mapsto fx_1, y \mapsto x_1\}}} (F \supset_l) \\
\frac{\overline{TP(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \overline{FP(z, x) \supset P(z, fx) \wedge P(x, ffy) \cdot \{y \mapsto x_1\}}}}{\overline{FP(z, x) \supset P(z, fx) \wedge P(x, ffy) \cdot \{x \mapsto ffx_1, y \mapsto x_2, z \mapsto fx_1\}, \overline{F \exists x \exists z (P(z, x) \supset P(z, fx) \wedge P(x, ffy)) \cdot \{y \mapsto x_1\}}}} (F \supset_l) \\
\frac{\overline{F \exists z (P(z, x) \supset P(z, fx) \wedge P(x, ffy)) \cdot \{x \mapsto ffx_1, y \mapsto x_2\}}, \overline{F \exists x \exists z (P(z, x) \supset P(z, fx) \wedge P(x, ffy)) \cdot \{y \mapsto x_1\}}}{\overline{F \exists x \exists z (P(z, x) \supset P(z, fx) \wedge P(x, ffy)) \cdot \{y \mapsto x_2\}}, \overline{F \exists x \exists z (P(z, x) \supset P(z, fx) \wedge P(x, ffy)) \cdot \{y \mapsto x_1\}}} (F \exists) \\
\frac{\overline{F \exists x \exists z (P(z, x) \supset P(z, fx) \wedge P(x, ffy)) \cdot \{y \mapsto x_2\}}, \overline{F \exists x \exists z (P(z, x) \supset P(z, fx) \wedge P(x, ffy)) \cdot \{y \mapsto x_1\}}}{\overline{F \forall y \exists x \exists z (P(z, x) \supset P(z, fx) \wedge P(x, ffy)) \cdot \epsilon}} (F \forall)
\end{array}$$

Figure 8: An C_{inv}^ϵ -derivation

- the last rule of Π is (C) ;
- the last rule of Π is $(F \wedge)$;
- the last rule of Π is $(F \vee_l)$;
- the last rule of Π is $(F \forall)$;
- the last rule of Π is $(F \exists)$.

At the beginning we note the following useful property of sequents in \mathbf{C}_{inv}^ξ -derivations.

For every rule of \mathbf{C}_{inv}^ξ if a pair $\alpha \cdot \tau$ is in the premise then $dom(\tau) = free(\alpha)$. Furthermore we can suppose that the same property holds for all pairs $\alpha \cdot \tau$ in sequents derived by induction hypothesis because of the renaming rule. (3.2)

Indeed, $dom(\tau) \subseteq free(\alpha)$ because we assume the substitution τ to be relevant for α . Likewise, $free(\alpha) \subseteq dom(\tau)$ because we require the premises to be variable-disjoint from $var(\xi)$, and therefore, from $free(\alpha)$.

In order to simplify the presentation we suppose that every sequent Γ in the rules of \mathbf{C}_{inv} , except for (Ax) and (C) consists only of one formula, and the sequents in (Ax) and (C) consist of at most two formulas.

CASE (Ax) . In this case the last inference has the form

$$\frac{}{T A_1 \theta_1, F A_2 \theta_2} (Ax),$$

where $T A_1, F A_2$ are free signed subformulas of ξ and $A_1 \theta_1 = A_2 \theta_2$. Without loss of generality we can assume that $dom(\theta_1) \subseteq free(A_1)$ ($dom(\theta_2) \subseteq free(A_2)$), otherwise we can take the substitutions $\theta_j \upharpoonright_{free(A_j)}$ instead of θ_j , for $j = 1, 2$. Let a renaming ρ rename A_1 away from A_2 . Since $A_1 \rho(\rho^{-1} \theta_1) = A_2 \theta_2$ we infer that $((\rho^{-1} \theta_1) \upharpoonright_{free(A_1 \rho)} \cup \theta_2)$ is a unifier of $A_1 \rho$ and A_2 . Therefore there exists a most general unifier σ of $A_1 \rho$ and A_2 such that $((\rho^{-1} \theta_1) \upharpoonright_{free(A_1 \rho)} \cup \theta_2) = \sigma \lambda$ for a substitution λ . We can conclude that the figure

$$\frac{}{T A_1 \cdot \rho \sigma, F A_2 \cdot \sigma} (Ax)$$

is the intended refutation in \mathbf{C}_{inv}^ξ . Indeed,

$$\begin{aligned} A_1 \theta_1 &= A_1 \rho((\rho^{-1} \theta_1) \upharpoonright_{free(A_1 \rho)} \cup \theta_2) = A_1 \rho \sigma \lambda, \\ A_2 \theta_2 &= A_2((\rho^{-1} \theta_1) \upharpoonright_{free(A_1 \rho)} \cup \theta_2) = A_2 \sigma \lambda. \end{aligned}$$

CASE (C) . The last inference in the \mathbf{C}_{inv} -refutation is

$$\frac{\alpha_1 \theta_1, \alpha_2 \theta_2}{\alpha_1 \theta_1} (C),$$

such that $\alpha_1 \theta_1 = \alpha_2 \theta_2$. By the induction hypothesis, there exist substitutions τ_1, τ_2, λ' such that $\alpha_j \theta_j = \alpha_j \tau_j \lambda'$ and the sequents $\alpha_j \cdot \tau_j$ are derivable in \mathbf{C}_{inv}^ξ , for

$j = 1, 2$. This implies that λ' is a unifier of $\alpha_1\theta_1$ and $\alpha_2\theta_2$. Therefore, there exists a most general unifier σ of $\alpha_1\theta_1$ and $\alpha_2\theta_2$ and a substitution λ such that $\sigma\lambda = \lambda'$. The following inference in \mathbf{C}_{inv}^ξ completes the proof of this case

$$\frac{\alpha_1 \cdot \tau_1, \alpha_2 \cdot \tau_2}{\alpha_1 \cdot \tau_1 \sigma} (C).$$

CASE $(F \wedge)$. The last inference in the \mathbf{C}_{inv} -refutation is

$$\frac{F A_1 \theta_1 \quad F A_2 \theta_2}{F (A_1 \wedge A_2) \theta} (F \wedge),$$

where, $\theta_j = \theta|_{free(A_j)}$, for $j = 1, 2$. By the induction hypothesis, there are sequents $F A_j \cdot \tau_j$ derivable in \mathbf{C}_{inv}^ξ and the substitutions λ_j such that $F A_j \theta_j = \alpha_j \tau_j \lambda_j$, for $j = 1, 2$. By using the renaming rule, we can assume $dom(\lambda_1) \cap dom(\lambda_2) = \emptyset$.

Let $\{x_1, \dots, x_n\} = dom(\tau_1) \setminus dom(\tau_2)$ and $\{y_1, \dots, y_m\} = dom(\tau_2) \setminus dom(\tau_1)$. Let $\lambda'_1 = \lambda_1 \cup \{y_1 \mapsto y_1 \tau_2 \lambda_2, \dots, y_m \mapsto y_m \tau_2 \lambda_2\}$ and $\lambda'_2 = \lambda_2 \cup \{x_1 \mapsto x_1 \tau_1 \lambda_1, \dots, x_n \mapsto x_n \tau_1 \lambda_1\}$. We can define a new substitution $\lambda' = \lambda'_1 \cup \lambda'_2$ and put it instead of λ_1 and λ_2 in the previous two equations. It results in the conclusion that λ' is a unifier of τ_1 and τ_2 because for every $x \in (dom(\tau_1) \cup dom(\tau_2))$ it holds that $x \tau_1 \lambda' = x \tau_2 \lambda' = x \theta$. Let $\sigma = mgu(\tau_1, \tau_2)$, then $\lambda' = \sigma \lambda$ for some substitution λ .

The following inference completes the proof of this case:

$$\frac{F A_1 \cdot \tau_1 \quad F A_2 \cdot \tau_2}{F A_1 \wedge A_2 \cdot \tau_1 \sigma} (F \wedge).$$

CASE $(F \vee_l)$. The last inference in the \mathbf{C}_{inv} -refutation is

$$\frac{F A \theta_1}{F (A_1 \vee A_2) \theta} (F \vee_l)$$

where $\theta_1 = \theta|_{free(A_1)}$.

By the induction hypothesis, there exist a sequent $F A_1 \cdot \tau_1$ refutable in \mathbf{C}_{inv}^ξ and a substitution λ' such that $F A_1 \theta_1 = F A \tau_1 \lambda'$. Let us show that the following inference completes the proof of this case:

$$\frac{F A_1 \cdot \tau_1}{F A_1 \vee A_2 \cdot \tau_1} (F \vee_l).$$

Let $\{x_1, \dots, x_n\} = dom(\theta) \setminus free(A)$. Define $\lambda = \lambda' \cup \{x_1 \mapsto x_1 \theta, \dots, x_n \mapsto x_n \theta\}$. Using $var(\tau_1) \cap var(A_2) = \emptyset$, it is not hard to argue that $(F A_1 \vee A_2) \tau \lambda = (F A_1 \vee A_2) \theta$.

CASE $(F \vee)$. The last inference of the \mathbf{C}_{inv} -refutation is

$$\frac{F A(y) \cdot \theta}{F \forall x A(x) \cdot \theta} (F \vee). \quad (3.3)$$

For simplicity, we only consider the case when $x \in \text{free}(A)$, the other case can be proved in a similar way. In this inference $x, y \notin \text{dom}(\theta)$, therefore the inference can be rewritten as

$$\frac{F A(x) \cdot \theta \cup \{x \mapsto y\}}{F \forall x A(x) \cdot \theta} (F \forall).$$

By the induction hypothesis, there exist substitutions τ, λ' and a sequent $F A(x) \cdot \tau$ refutable in \mathbf{C}_{inv}^ξ such that

$$A(x)\tau\lambda' = A(x)\theta \cup \{x \mapsto y\}. \quad (3.4)$$

To conclude the proof of this case, it is enough to find a substitution λ such that

$$\forall x A(x)\tau_{-x}\lambda = \forall x A(x)\theta \quad (3.5)$$

and

$$\frac{F A(x) \cdot \tau}{F \forall x A(x) \cdot \tau_{-x}} (F \forall) \quad (3.6)$$

is a valid inference in \mathbf{C}_{inv}^ξ .

We prove (3.5) by letting $\lambda = \lambda'$. Indeed, since $x \notin \text{free}(\forall x A(x))$, we have $\forall x A(x)\tau_{-x}\lambda = \forall x A(x)\tau\lambda$. Condition (3.4) implies $\forall x A(x)\tau_{-x}\lambda = \forall x A(x)\theta \cup \{x \mapsto y\}$. Using $x \notin \text{dom}(\theta)$, we obtain (3.5).

To prove that (3.6) is a valid inference in \mathbf{C}_{inv}^ξ , it is enough to check the eigenvariable condition: $x\tau$ is a variable not occurring in $\forall x A(x)\tau_{-x}$. Condition (3.4) implies $x\tau\lambda' = x\theta \cup \{x \mapsto y\}$. Since $x \notin \text{dom}(\theta)$, we have $x\tau\lambda' = y$. Therefore, $x\tau$ is a variable, denote this variable by z . Note that $z\lambda' = y$. To check the eigenvariable condition suppose, by contradiction, $z \in \text{free}(\forall x A(x)\tau_{-x})$. Then $z\lambda' \in \text{free}(\forall x A(x)\tau_{-x}\lambda')$. From $z\lambda' = y$ it follows that $y \in \text{free}(\forall x A(x)\tau_{-x}\lambda')$. Equation (3.5) yields $y \in \text{free}(\forall x A(x)\theta)$. This contradicts the eigenvariable condition for inference (3.3).

CASE (T \forall). This case is quite similar to the previous one, except that we do not need to verify the troublesome eigenvariable condition. □

To establish the soundness of \mathbf{C}_{inv}^ξ , one can prove the following lemma.

3.17. LEMMA. *If a sequent $\alpha_1 \cdot \theta_1, \dots, \alpha_n \cdot \theta_n$ has a refutation in \mathbf{C}_{inv}^ξ , then the sequent $\alpha_1\theta_1, \dots, \alpha_n\theta_n$ has a refutation in \mathbf{C}_{inv} .* □

3.18. THEOREM (Completeness of \mathbf{C}_{inv}^ξ). *Let ξ be a closed signed formula. Then ξ is unsatisfiable if and only if ξ has a refutation in \mathbf{C}_{inv}^ξ .*

PROOF. (\Rightarrow). Suppose that ξ is unsatisfiable. By Completeness Theorem 3.18 for \mathbf{C}_{inv} , ξ has a refutation in \mathbf{C}_{inv} . By Instance Lemma 3.16, ξ has a refutation in \mathbf{C}_{inv}^ξ .

(\Leftarrow). Suppose that ξ has a refutation in \mathbf{C}_{inv}^ξ . By Lemma 3.17, ξ has a refutation in \mathbf{C}_{inv} . By Completeness Theorem 3.18 for \mathbf{C}_{inv} , ξ is unsatisfiable. \square

It follows from this theorem that, for any closed formula A , the sequent $F A$ has a refutation in \mathbf{C}_{inv}^ξ if and only if A is valid.

Using unifiable atoms instead of weakly unifiable in the axiom (Ax) gives an incomplete calculus. Consider, for example, the signed formula $\xi = F \exists x(P(x) \supset P(fx))$. To apply the rule (Ax) to $P(x)$ and $P(fx)$, we take $\rho = \{x \mapsto y\}$ and use the most general unifier $\{y \mapsto fx\}$ of $P(y)$ and $P(fx)$:

$$T P(x) \cdot \{x \mapsto fx\}, F P(fx) \cdot \varepsilon.$$

This axiom can be extended to a refutation of the sequent. However, if we use unifiability instead of weak unifiability, no axiom can be built since $P(x)$ and $P(fx)$ are not unifiable.¹

3.4. Negation normal forms

In order to decrease the number of rules and simplify the presentation, we will consider formulas of a special form, called the negation normal form. In classical logic and all modal logics discussed in this paper every formula is equivalent to a formula in negation normal form.

3.19. DEFINITION (*literal, negation normal form*). A **literal** is an atomic formula or its negation. For every literal L , the literal **complementary** to L , denoted \bar{L} is defined as follows:

$$\bar{L} = \begin{cases} A, & \text{if } L = \neg A; \\ \neg L, & \text{otherwise.} \end{cases}$$

A formula is in **negation normal form**, or simply in **NNF**, if it is constructed from literals using \wedge , \vee , \forall and \exists . A **negation normal form of a formula** A is any formula in negation normal form equivalent to A .

We will now change the definition of signed subformulas into a definition of subformulas of NNFs.

3.20. DEFINITION (*subformula*). The notion of a **free subformula** of a NNF A is defined by induction as follows. First, we define **free immediate subformulas** and **immediate subformulas** using the table of Figure 6.

¹Tammet [1997] uses unifiability instead of weak unifiability in the axioms for intuitionistic logic. It seems that he assumes an implicit variable renaming rule. For intuitionistic logic, a similar counterexample to completeness is $\neg\neg\exists x(P(x) \supset P(fx))$.

In the table below, t is an arbitrary term free for x in $A(x)$.

formula	its free immediate	its immediate
	subformulas	subformulas
$A \wedge B$	A and B	A and B
$A \vee B$	A and B	A and B
$\exists x A(x)$	$A(x)$	$A(t)$
$\forall x A(x)$	$A(x)$	$A(t)$

Figure 9: Immediate subformulas

For all formulas A, B, C we define various notions of subformulas as follows.

1. A is a (free) signed subformula of A ;
2. If A is a (free) immediate subformula of B and B is a (free) subformula of C , then A is a (free) subformula of C .

Note that a formula $\neg A$ has no immediate subformulas, and therefore the only subformula of $\neg A$ is $\neg A$ itself.

It is known that every formula can be transformed in its negation normal form using the following transformations:

$$\begin{aligned}
 A \supset B &\Rightarrow \neg A \vee B, \\
 \neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
 \neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
 \neg\neg A &\Rightarrow A, \\
 \neg\exists x A &\Rightarrow \forall x \neg A, \\
 \neg\forall x A &\Rightarrow \exists x \neg A.
 \end{aligned}$$

When we have a formula A in negation normal form and try to refute $\text{T } A$, the calculus \mathbf{C}_{inv} can be considerably simplified. Essentially, we should only be dealing with signed formulas of the form $\text{T } B$, where B is in negation normal form. Since there is only one sign, there is no reason to consider signs at all, so sequents can be viewed as multisets of formulas. A multiset of formulas B_1, \dots, B_n corresponds to the multiset of signed formulas $\text{T } B_1, \dots, \text{T } B_n$. Instead of a goal signed formula $\xi = \text{T } G$, we will deal with the goal formula G .

CONVENTION In the sequel G denotes the goal formula in NNF.

The calculi \mathbf{C}_{tab} , \mathbf{C}_{inv} and \mathbf{C}_{inv}^ξ specialized to formulas in NNF are shown in Figures 10–12. Since we now deal with a goal formula G instead of a goal signed formula ξ , the calculus \mathbf{C}_{inv}^ξ is denoted by \mathbf{C}_{inv}^G . We do not give conditions on the inference rules, the reader can derive these conditions from the conditions on the original calculi.

$$\begin{array}{c}
\frac{}{\Gamma, A, \neg A} (Ax) \qquad \frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \vee B} (\vee) \qquad \frac{\Gamma, A, B}{\Gamma, A \wedge B} (\wedge) \\
\frac{\Gamma, A(y)}{\Gamma, \exists x A(x)} (\exists) \qquad \frac{\Gamma, \forall x A(x), A(t)}{\Gamma, \forall x A(x)} (\forall)
\end{array}$$

Figure 10: Calculus C_{tab} for formulas in NNF

$$\begin{array}{c}
\frac{}{A, \neg A} (Ax) \qquad \frac{\Gamma, A, A}{\Gamma, A} (C) \\
\frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \vee B} (\vee) \qquad \frac{\Gamma, A}{\Gamma, A \wedge B} (\wedge_l) \qquad \frac{\Gamma, B}{\Gamma, A \wedge B} (\wedge_r) \\
\frac{\Gamma, A(y)}{\Gamma, \exists x A(x)} (\exists) \qquad \frac{\Gamma, A(t)}{\Gamma, \forall x A(x)} (\forall)
\end{array}$$

Figure 11: Calculus C_{inv} for formulas in NNF

$$\begin{array}{c}
\frac{}{A \cdot \rho\tau, \neg B \cdot \tau} (Ax) \qquad \frac{\Gamma, A \cdot \sigma_1, A \cdot \sigma_2}{\Gamma\theta, A \cdot \sigma_1\theta} (C) \\
\frac{\Gamma, A \cdot \sigma_1 \quad \Delta, B \cdot \sigma_2}{\Gamma\theta, \Delta\theta, A \vee B \cdot \sigma_1\theta} (\vee) \\
\frac{\Gamma, A \cdot \sigma}{\Gamma, A \wedge B \cdot \sigma} (\wedge_l) \qquad \frac{\Gamma, B \cdot \sigma}{\Gamma, A \wedge B \cdot \sigma} (\wedge_r) \\
\frac{\Gamma, A \cdot \sigma}{\Gamma, \exists x A \cdot \sigma_{-x}} (\exists) \qquad \frac{\Gamma, A \cdot \sigma}{\Gamma, \forall x A \cdot \sigma_{-x}} (\forall)
\end{array}$$

Figure 12: Calculus C_{inv}^G for formulas in NNF

3.21. **EXAMPLE.** Consider again signed formula (3.1) on page 192. The corresponding formula in negation normal form is

$$G = \exists y \forall x \forall z (P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, fgy))). \quad (3.7)$$

Using the calculus C_{inv}^G for formulas in negation normal form, the derivation of (3.1) of Figure 8 can be turned into the derivation of this formula shown in Figure 13.

3.5. Saturation algorithms and linear representation of derivations

Proof-search by the inverse method is based on a **saturation algorithm**. This algorithm makes iterative computations over a database of sequents. Initially, the database contains all axioms of C_{inv}^G . At every iteration step an inference or set of inferences is applied to one or more sequents in the database. The conclusions of these inferences are added to the database. The proof-search terminates if the goal is in the database, or if no new sequents can be added. Sometimes, **redundancy criteria** are applied to the database. Each redundancy criterion identifies sequents that are redundant and can be removed from the database. There are also redundancy criteria for inferences. Redundancy criteria for the inverse method are discussed in Section 6. Saturation-based algorithms and redundancies for resolution are discussed in [Bachmair and Ganzinger 2001] (Chapter 2 of this Handbook), for paramodulation-based reasoning in [Nieuwenhuis and Rubio 2001] (Chapter 7), and from the practical viewpoint in [Weidenbach 2001] (Chapter 27).

In such saturation algorithms, the same sequents can be reused over and over again. Typical derivation trees can be tremendously large (in resolution-based inference systems more than 2^{50} clauses), while the number of sequents used in the derivation is typically small (no more than a few thousand). Therefore, we obtain a more economical representation of derivations by only displaying the sequents instead of derivation trees. We will call them **linear representations of derivations**.²

In the sequel, we will only use linear representations of derivations, with numbered sequents. With every sequent, we will write the name of the inference rule used to obtain this sequent, and the numbers of its parents: the sequents in the premises of the inference used to obtain this sequent. The linear representation of the derivation of Figure 13 is shown in Figure 14.

3.6. Exploiting other properties of sequent calculi

What we have achieved in this section is the construction of the calculus C_{inv}^G that can be used to derive formulas in an automatic way. To make a further step towards proof-search, we have to introduce some **redundancy criteria** to restrict the

²A *linear derivation* would be an appropriate term for such kind of derivations, however, this term resembles linear formats for resolution (see e.g., [Kowalski and Kuehner 1971]).

$$\begin{array}{c}
\frac{P(z, x) \cdot \{x \mapsto fx_2, z \mapsto x_1\}, \neg P(z, fx) \cdot \{x \mapsto x_2, z \mapsto x_1\}}{P(z, x) \cdot \{x \mapsto fx_2, z \mapsto x_1\}, P(z, x) \cdot \{x \mapsto ffx_3, z \mapsto x_2\}, \neg P(z, fx) \vee \neg P(x, ffx) \cdot \{x \mapsto x_2, y \mapsto x_3, z \mapsto x_1\}} \quad (\vee) \\
\frac{P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \neg P(z, fx) \vee \neg P(x, ffx) \cdot \{x \mapsto x_2, y \mapsto x_3, z \mapsto x_1\}}{P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, P(z, x) \cdot \{x \mapsto ffx_3, z \mapsto x_2\}, \neg P(z, fx) \vee \neg P(x, ffx) \cdot \{x \mapsto x_2, y \mapsto x_3, z \mapsto x_1\}} \quad (C) \\
\frac{P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \neg P(z, fx) \vee \neg P(x, ffx) \cdot \{x \mapsto x_2, y \mapsto x_3, z \mapsto x_1\}}{P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, P(z, x) \cdot \{x \mapsto ffx_3, z \mapsto x_2\}, \neg P(z, fx) \vee \neg P(x, ffx) \cdot \{x \mapsto x_2, y \mapsto x_3, z \mapsto x_1\}} \quad (\wedge_r) \\
\frac{P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \neg P(z, fx) \vee \neg P(x, ffx) \cdot \{x \mapsto x_2, y \mapsto x_3, z \mapsto x_1\}}{P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx))) \cdot \{x \mapsto ffx_1, y \mapsto x_1\}} \quad (\forall) \\
\frac{P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx))) \cdot \{x \mapsto ffx_1, y \mapsto x_1\}}{P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\}, \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx))) \cdot \{y \mapsto x_1\}} \quad (\forall) \\
\frac{P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx)) \cdot \{x \mapsto ffx_1, y \mapsto x_2, z \mapsto fx_1\}, \forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx))) \cdot \{y \mapsto x_1\}}{P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx)) \cdot \{x \mapsto ffx_1, y \mapsto x_2\}, \forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx))) \cdot \{y \mapsto x_1\}} \quad (\wedge_l) \\
\frac{\forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx))) \cdot \{y \mapsto x_2\}, \forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx))) \cdot \{y \mapsto x_1\}}{\forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx))) \cdot \{y \mapsto x_1\}} \quad (\forall) \\
\frac{\forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx))) \cdot \{y \mapsto x_1\}}{\exists y \forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffx))) \cdot \varepsilon} \quad (\exists)
\end{array}$$

Figure 13: An C^G_{inv} -derivation of a formula in negation normal form

- (1) $P(z, x) \cdot \{x \mapsto fx_2, z \mapsto x_1\},$
 $\neg P(z, fx) \cdot \{x \mapsto x_2, z \mapsto x_1\}$ axiom
- (2) $P(z, x) \cdot \{x \mapsto ffy_2, z \mapsto y_1\},$
 $\neg P(x, ffy) \cdot \{x \mapsto y_1, y \mapsto y_2\}$ axiom
- (3) $P(z, x) \cdot \{x \mapsto fx_2, z \mapsto x_1\},$
 $P(z, x) \cdot \{x \mapsto ffx_3, z \mapsto x_2\},$
 $\neg P(z, fx) \vee \neg P(x, ffy) \cdot \{x \mapsto x_2, y \mapsto x_3, z \mapsto x_1\}$ (V) from (1,2)
- (4) $P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\},$
 $\neg P(z, fx) \vee \neg P(x, ffy) \cdot \{x \mapsto fx_1, y \mapsto x_1, z \mapsto fx_1\}$ (C) from (3)
- (5) $P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\},$
 $P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))$
 $\cdot \{x \mapsto fx_1, y \mapsto x_1, z \mapsto fx_1\}$ (\wedge_r) from (4)
- (6) $P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\},$
 $\forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))) \cdot \{x \mapsto fx_1, y \mapsto x_1\}$ (V) from (5)
- (7) $P(z, x) \cdot \{x \mapsto ffx_1, z \mapsto fx_1\},$
 $\forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))) \cdot \{y \mapsto x_1\}$ (V) from (6)
- (8) $P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))$
 $\cdot \{x \mapsto ffx_1, y \mapsto x_2, z \mapsto fx_1\},$
 $\forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))) \cdot \{y \mapsto x_1\}$ (\wedge_l) from (7)
- (9) $\forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))) \cdot \{x \mapsto ffx_1, y \mapsto x_2\},$
 $\forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))) \cdot \{y \mapsto x_1\}$ (V) from (8)
- (10) $\forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))) \cdot \{y \mapsto x_2\},$
 $\forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))) \cdot \{y \mapsto x_1\}$ (V) from (9)
- (11) $\forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))) \cdot \{y \mapsto x_1\}$ (C) from (10)
- (12) $\exists y \forall x \forall z(P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))) \cdot \varepsilon$ (\exists) from (11)

Figure 14: The linear representation of the derivation of Figure 13

search space, similar to those extensively studied in [Bachmair and Ganzinger 2001] (Chapter 2 of this Handbook). There are essentially two kinds of redundancies in proof-search systems: redundant sequents and redundant derivations. For the inverse method, some of these criteria (like subsumption) can be taken over from resolution or from other very general concepts [Maslov 1983c, Voronkov 1992, Mints, Orevkov and Tammet 1996]. Other criteria are based on properties of sequent calculi. We will consider redundancy criteria in Section 6.

Another important aspect is the representation of sequents. In Section 5 we will consider a naming technique that allows one to consider clauses instead of sequents. In the case of classical logic, the naming technique augmented with the treatment of inference rules as clauses gives us the simulation of the inverse method by hyperresolution [Maslov 1969, Maslov 1971b, Kuechner 1971], see also [Bachmair and

Ganzinger 2001, page 76] (Chapter 2 of this volume).

4. Applying the recipe to nonclassical logics

In this section we apply the universal recipe to cook the inverse method calculi for several nonclassical logics: intuitionistic logic and some modal logics. There are only a few papers on the inverse method for nonclassical logics, and some authors characterize the inverse method as “resolution” for nonclassical logic due to the close relationships between resolution derivations and the inverse method derivations explained in Section 5.2. We avoid calling the inverse method “resolution” for the following reasons.

1. The inverse method is not regarded as resolution for classical logic, where there is a bisimulation of hyperresolution derivations and the inverse method derivations.
2. The inverse method systems for nonclassical logics contain inference rules that cannot be simulated by the standard resolution rule. As a consequence, the “resolution” calculi for nonclassical logics contain “clauses” encoding these rules for which special non-resolution inference rules are introduced. There is no semantics behind these rules unless we translate them back to the sequent calculus rules.

The inverse method for nonclassical logics is discussed in [Mints 1990, Auffray, Enjalbert and Hebrard 1990, Mints et al. 1996, Voronkov 2000a, Voronkov 2001, Voronkov 2000b] for modal logics, in [Voronkov 1992] for a number of logics, including intuitionistic logic, modal logics and logics without the contraction rule, in [Mints 1993b, Tammet 1994] for linear logic, and in [Mints 1994, Tammet 1996, Tammet 1997] for intuitionistic logic.

4.1. Intuitionistic logic

In the case of intuitionistic logic, the design of a suitable inverse method calculus follows the same recipe as for classical logic. We begin with a cut-free calculus I_{tab} , change it into an inverse method calculus, and finally use the subformula property to add free variables to it.

We call an *intuitionistic sequent* any sequent with at most one signed formula of the form $F A$. The *sequent calculus for intuitionistic logic* I_{tab} is the calculus of intuitionistic sequents shown in Figure 15. This calculus is essentially $G3$ of Kleene [1967, page 481].

To find the right rules for the inverse method calculus out of I_{tab} , we have to adapt the proof of Subsequent Lemma 3.7 to the intuitionistic case. This gives us the calculus I_{inv} shown in Figure 16. Let us explain why some rules of the system look different from their classical counterpart, even when the tableau rules are the same. A typical example is the rule (TV) . For classical logic, the inverse method rule has the form

$$\begin{array}{c}
\overline{\Gamma, \top A, \bot A} \quad (Ax) \\
\\
\frac{\Gamma, \top A, \top B}{\Gamma, \top A \wedge B} \quad (T \wedge) \qquad \frac{\Gamma, \bot A \quad \Gamma, \bot B}{\Gamma, \bot A \wedge B} \quad (F \wedge) \\
\\
\frac{\Gamma, \top A \quad \Gamma, \top B}{\Gamma, \top A \vee B} \quad (T \vee) \qquad \frac{\Gamma, \bot A}{\Gamma, \bot A \vee B} \quad (F \vee_l) \qquad \frac{\Gamma, \bot B}{\Gamma, \bot A \vee B} \quad (F \vee_r) \\
\\
\frac{\Gamma, \top A \supset B, \bot A \quad \Gamma, \top B}{\Gamma, \top A \supset B} \quad (T \supset) \qquad \frac{\Gamma, \top A, \bot B}{\Gamma, \bot A \supset B} \quad (F \supset) \\
\\
\frac{\Gamma, \top \neg A, \bot A}{\Gamma, \top \neg A} \quad (T \neg_1) \qquad \frac{\Gamma, \top \neg A, \bot A}{\Gamma, \top \neg A, \bot B} \quad (T \neg_2) \qquad \frac{\Gamma, \top A}{\Gamma, \bot \neg A} \quad (F \neg) \\
\\
\frac{\Gamma, \top \forall x A(x), \top A(t)}{\Gamma, \top \forall x A(x)} \quad (T \forall) \qquad \frac{\Gamma, \bot A(y)}{\Gamma, \bot \forall x A(x)} \quad (F \forall) \\
\\
\frac{\Gamma, \top A(y)}{\Gamma, \top \exists x A(x)} \quad (T \exists) \qquad \frac{\Gamma, \bot A(t)}{\Gamma, \bot \exists x A(x)} \quad (F \exists)
\end{array}$$

In the rule (Ax) , A is an atomic formula. The rules $(F \forall)$ and $(T \exists)$ satisfy the eigenvariable condition.

Figure 15: Calculus I_{tab}

$$\frac{\Gamma, \top A \quad \Delta, \top B}{\Gamma, \Delta, \top A \vee B} \quad (T \vee).$$

In order to obtain the sequent $\top A \vee B, FC$ from $\top A, FC$ and $\top B, FC$ in classical logic, we can use the following derivation

$$\frac{\frac{\top A, FC \quad \top B, FC}{\top A \vee B, FC, FC} \quad (T \vee)}{\top A \vee B, FC} \quad (C). \tag{4.1}$$

In the intuitionistic system, this derivation cannot be used, since $\top A \vee B, FC, FC$ is not a valid sequent. To avoid this problem, we augment the intuitionistic inverse method system I_{inv} with an additional rule

$$\frac{\Gamma, \top A, FC \quad \Delta, \top B, FC}{\Gamma, \Delta, \top A \vee B, FC} \quad (T \vee_2).$$

The problem we had with the rule $(T \vee)$ appears due to the impossibility of applying contraction to a signed formula (FC) : such an application would require two

$$\begin{array}{c}
\frac{}{\top A, \text{FA}} (Ax) \\
\frac{\Gamma, \top A, \top A}{\Gamma, \top A} (C) \\
\frac{\Gamma, \top A}{\Gamma, \top A \wedge B} (\top \wedge) \qquad \frac{\Gamma, \top B}{\Gamma, \top A \wedge B} (\top \wedge) \\
\frac{\Gamma, \text{FA} \quad \Delta, \text{FB}}{\Gamma, \Delta, \text{FA} \wedge B} (\text{F} \wedge) \\
\frac{\Gamma, \top A \quad \Delta, \top B}{\Gamma, \Delta, \top A \vee B} (\top \vee_1) \qquad \frac{\Gamma, \top A, \text{FC} \quad \Delta, \top B, \text{FC}}{\Gamma, \Delta, \top A \vee B, \text{FC}} (\top \vee_2) \\
\frac{\Gamma, \text{FA}}{\Gamma, \text{FA} \vee B} (\text{F} \vee_l) \qquad \frac{\Gamma, \text{FB}}{\Gamma, \text{FA} \vee B} (\text{F} \vee_r) \\
\frac{\Gamma, \text{FA} \quad \Delta, \top B}{\Gamma, \Delta, \top A \supset B} (\top \supset) \\
\frac{\Gamma, \top A}{\Gamma, \text{FA} \supset B} (\text{F} \supset_l) \qquad \frac{\Gamma, \text{FB}}{\Gamma, \text{FA} \supset B} (\text{F} \supset_r) \\
\frac{\Gamma, \top A, \text{FA} \supset B}{\Gamma, \text{FA} \supset B} (\text{F} \supset_{lr}) \\
\frac{\Gamma, \text{FA}}{\Gamma, \top \neg A} (\top \neg) \qquad \frac{\Gamma, \top A}{\Gamma, \text{F} \neg A} (\text{F} \neg) \\
\frac{\Gamma, \top A(t)}{\Gamma, \top \forall x A(x)} (\top \forall) \qquad \frac{\Gamma, \text{FA}(y)}{\Gamma, \text{F} \forall x A(x)} (\text{F} \forall) \\
\frac{\Gamma, \top A(y)}{\Gamma, \top \exists x A(x)} (\top \exists) \qquad \frac{\Gamma, \text{FA}(t)}{\Gamma, \text{F} \exists x A(x)} (\text{F} \exists)
\end{array}$$

In the rule (Ax) , A is an atomic formula. The rules $(\text{F}\forall)$ and $(\text{F}\exists)$ satisfy the eigenvariable condition.

Figure 16: Calculus \mathbf{I}_{inv}

copies of $(F C)$ in a sequent, which is prohibited by the definition of intuitionistic sequents. The same problem arises in connection with the rule $(F \supset)$. This problem was overlooked in [Tammet 1996, Tammet 1997] where an incorrect recipe for intuitionistic logic is given.

4.1. LEMMA. *Let Γ have a refutation in I_{tab} . Then there exists a sequent Δ such that $\Delta \subseteq \Gamma$ and Δ has a refutation in I_{inv} .*

PROOF. The proof is similar to that of Lemma 3.7. We only consider some cases where the rules for intuitionistic logic are substantially different from those for classical logic. These are $(T \supset)$, $(T \neg)$ (the rules of I_{tab} are quite different from those of C_{tab}) and $(F \supset)$ (the rules of I_{inv} are quite different from those of C_{inv}). **CASE $(T \supset)$.** The rule of I_{tab} has the form

$$\frac{\Gamma, T A \supset B, F A \quad \Gamma, T B}{\Gamma, T A \supset B} (T \supset).$$

By the induction hypothesis, there exist sequents $\Delta_1 \subseteq (\Gamma, T A \supset B, F A)$ and $\Delta_2 \subseteq (\Gamma, T B)$ is refutable in I_{inv} . We have to find a sequent $\Delta \subseteq (\Gamma, T A \supset B)$ refutable in I_{inv} .

If we have $\Delta_1 \subseteq (\Gamma, T A \supset B)$, then we can take Δ_1 to be Δ . Likewise, if we have $\Delta_2 \subseteq \Gamma$, then we can take Δ_2 to be Δ . So we can assume that $(F A) \in \Delta_1$ and $(T B) \in \Delta_2$. We will only consider the case when $T A \supset B$ occurs in Δ_1 , leaving the remaining case to the reader. In this case we can assume that $\Delta_1 = (\Delta'_1, T A \supset B, F A)$ and $\Delta_2 = (\Delta'_2, T B)$ for some $\Delta'_1 \subseteq \Gamma$ and $\Delta'_2 \subseteq \Gamma$. Since $\Delta'_1 \subseteq \Gamma$ and $\Delta'_2 \subseteq \Gamma$, and Γ contains no formulas of the form $F C$, some sequent $\Gamma' \subseteq \Gamma$ can be obtained from the sequent Δ'_1, Δ'_2 in I_{inv} by a series of contractions. The following derivation in I_{inv} proves this case:

$$\begin{array}{c} \frac{\Delta'_1, T A \supset B, F A \quad \Delta'_2, T B}{\Delta'_1, \Delta'_2, T A \supset B, T A \supset B} (T \supset) \\ \vdots \text{ series of contractions} \\ \Gamma', T A \supset B \end{array}$$

CASE $(F \supset)$. This case gives rise to three rules of I_{inv} . The $(F \supset)$ rule of I_{tab} has the form

$$\frac{\Gamma, T A, F B}{\Gamma, F A \supset B} (F \supset).$$

By the induction hypothesis, there exists a sequent $\Delta \subseteq (\Gamma, T A, T B)$ refutable in I_{inv} . We have to find a sequent $\Delta' \subseteq (\Gamma, F A \supset B)$ refutable in I_{inv} . Consider the following cases, depending on whether $T A$ and $F B$ occur in Δ .

SUBCASE $((T A, F B) \subseteq \Delta)$. We have $\Delta = (\nabla, T A, F B)$, for some $\nabla \subseteq \Gamma$. The following derivation in I_{inv} proves this case:

$$\frac{\nabla, T A, F B}{\nabla, T A, F A \supset B} (F \supset_r) \\ \frac{\nabla, T A, F A \supset B}{\nabla, F A \supset B} (F \supset_{lr}).$$

SUBCASE ($\nabla A \in \Delta$ and $\Delta \dot{\subseteq} (\Gamma, \nabla A)$). We have $\Delta = \nabla, \nabla A$, for some $\nabla \dot{\subseteq} \Gamma$. The following derivation in \mathbf{I}_{inv} proves this case:

$$\frac{\nabla, \nabla A}{\nabla, \nabla A \supset B} (F \supset_l).$$

SUBCASE ($\nabla B \in \Delta$ and $\Delta \dot{\subseteq} (\Gamma, \nabla B)$). This case is similar to the previous one.

SUBCASE ($\Delta \dot{\subseteq} \Gamma$). In this case we let $\Delta' = \Delta$.

CASE ($\nabla \neg$). This case is similar to ($F \supset$), but requires an extra contraction by ($\nabla \neg A$), like in the case ($\nabla \supset$). We leave the details to the reader. □

As we can see from the proof, the application of one part of the recipe to intuitionistic logic gives us a few surprises. Two rules for ($\nabla \neg$) in \mathbf{I}_{tab} become one rule in \mathbf{I}_{inv} , while one rule for ($F \supset$) in \mathbf{I}_{tab} becomes three rules in \mathbf{I}_{inv} . We could have obtained a smaller number of rules had we used sets instead of multisets. However, the use of sets would have caused a problem with lifting, resulting in the same free-variable system, and in a more complex proof of completeness for the free-variable system.

Before defining the calculus \mathbf{I}_{inv}^ξ we have to generalize the notion of most general unifier of substitutions to two pairs of substitutions. A **simultaneous most general unifier of the pairs of substitutions** σ_1, τ_1 and σ_2, τ_2 is a substitution ϑ defined as follows. Let $\{x_1, \dots, x_n\} = \text{dom}(\sigma_1) \cup \text{dom}(\sigma_2)$ and $\{y_1, \dots, y_m\} = \text{dom}(\tau_1) \cup \text{dom}(\tau_2)$, $n, m \geq 0$. Then ϑ is any most general unifier of the tuples $\langle x_1\sigma_1, \dots, x_n\sigma_1, y_1\tau_1, \dots, y_m\tau_1 \rangle$ and $\langle x_1\sigma_2, \dots, x_n\sigma_2, y_1\tau_2, \dots, y_m\tau_2 \rangle$. The idea of this definition is that we have to unify σ_1 with σ_2 and also τ_1 with τ_2 *simultaneously*.

The free-variable system \mathbf{I}_{inv}^ξ is given in Figure 17.

Again, by comparing \mathbf{I}_{inv} and \mathbf{I}_{inv}^ξ we obtain the same lifting arguments as for \mathbf{C}_{inv} and \mathbf{C}_{inv}^ξ .

4.2. LEMMA (Instance Lemma for \mathbf{I}_{inv}^ξ). *Let ξ be a closed signed formula and $\alpha_1, \dots, \alpha_n$ be free signed subformulas of ξ . Let $\theta_1, \dots, \theta_n$ be substitutions such that the sequent $\alpha_1\theta_1, \dots, \alpha_n\theta_n$ is derivable in \mathbf{I}_{inv} . Then there exist substitutions $\lambda, \tau_1, \dots, \tau_n$ such that the sequent $\alpha_1 \cdot \tau_1, \dots, \alpha_n \cdot \tau_n$ is derivable in \mathbf{I}_{inv}^ξ and $\alpha_i\tau_i\lambda = \alpha_i\theta_i$ for all $i \in \{1, \dots, n\}$.*

PROOF. The proof is quite similar to the proof of Lemma 3.16 above. In view of that proof, we will only consider the induction step in the case when the last inference of the derivation Π in \mathbf{I}_{inv} is (∇V_2). Derivation 4.1 on page 210 demonstrates that any inference by this rule in \mathbf{I}_{inv} is equivalent to a sequence of inferences by (∇V) and (C) in \mathbf{C}_{inv} . We can also see that the rules (∇V) and (C) in \mathbf{C}_{inv}^ξ have the same form as (∇V_1) and (C) in \mathbf{I}_{inv}^ξ , respectively. Because the Instance Lemma is valid for \mathbf{C}_{inv}^ξ , it follows that we have to prove only that a sequence of two inferences by

In the rule (Ax) below, A and B are weakly unifiable atoms, ρ renames away A from B and τ is a most general unifier of $A\rho$ and B . In all rules the premises are variable-disjoint. In all rules each premise is also variable-disjoint with $\text{var}(\xi)$. In all rules θ is a most general unifier of σ_1, σ_2 . In the rule $(T\vee_2)$ ϑ is a simultaneous most general unifier of the pairs σ_1, δ_1 and σ_2, δ_2 . The rules $(F\vee)$ and $(T\exists)$ satisfy the eigenvariable condition: the term $x\sigma$ is a variable not occurring in the conclusion of the rule.

$$\begin{array}{c}
\frac{}{\text{TA} \cdot \rho\tau, \text{FB} \cdot \tau} (Ax) \\
\frac{\Gamma, \text{TA} \cdot \sigma_1, \text{TA} \cdot \sigma_2}{\Gamma\theta, \text{TA} \cdot \sigma_1\theta} (C) \\
\frac{\Gamma, \text{TA} \cdot \sigma}{\Gamma, \text{TA} \wedge B \cdot \sigma} (T\wedge_l) \qquad \frac{\Gamma, \text{TB} \cdot \sigma}{\Gamma, \text{TA} \wedge B \cdot \sigma} (T\wedge_r) \\
\frac{\Gamma, \text{FA} \cdot \sigma_1 \quad \Delta, \text{FB} \cdot \sigma_2}{\Gamma\theta, \Delta\theta, \text{FA} \wedge B \cdot \sigma_1\theta} (F\wedge) \\
\frac{\Gamma, \text{TA} \cdot \sigma_1 \quad \Delta, \text{TB} \cdot \sigma_2}{\Gamma\theta, \Delta\theta, \text{TA} \vee B \cdot \sigma_1\theta} (T\vee_1) \\
\frac{\Gamma, \text{TA} \cdot \sigma_1, \text{FC} \cdot \delta_1 \quad \Delta, \text{TB} \cdot \sigma_2, \text{FC} \cdot \delta_2}{\Gamma\vartheta, \Delta\vartheta, \text{TA} \vee B \cdot \sigma_1\vartheta, \text{FC} \cdot \delta_1\vartheta} (T\vee_2) \\
\frac{\Gamma, \text{FA} \cdot \sigma}{\Gamma, \text{FA} \vee B \cdot \sigma} (F\vee_l) \qquad \frac{\Gamma, \text{FB} \cdot \sigma}{\Gamma, \text{FA} \vee B \cdot \sigma} (F\vee_r) \\
\frac{\Gamma, \text{FA} \cdot \sigma_1 \quad \Delta, \text{TB} \cdot \sigma_2}{\Gamma\theta, \Delta\theta, \text{TA} \supset B \cdot \sigma_1\theta} (T\supset) \\
\frac{\Gamma, \text{TA} \cdot \sigma}{\Gamma, \text{FA} \supset B \cdot \sigma} (F\supset_l) \qquad \frac{\Gamma, \text{FB} \cdot \sigma}{\Gamma, \text{FA} \supset B \cdot \sigma} (F\supset_r) \\
\frac{\Gamma, \text{TA} \cdot \sigma_1, \text{FA} \supset B \cdot \sigma_2}{\Gamma\theta, \text{FA} \supset B \cdot \sigma_2\theta} (F\supset_{lr}) \\
\frac{\Gamma, \text{FA} \cdot \sigma}{\Gamma, \text{T}\neg A \cdot \sigma} (T\neg) \qquad \frac{\Gamma, \text{TA} \cdot \sigma}{\Gamma, \text{F}\neg A \cdot \sigma} (F\neg) \\
\frac{\Gamma, \text{TA} \cdot \sigma}{\Gamma, \text{T}\forall xA \cdot \sigma_{-x}} (T\forall) \qquad \frac{\Gamma, \text{FA} \cdot \sigma}{\Gamma, \text{F}\forall xA \cdot \sigma_{-x}} (F\forall) \\
\frac{\Gamma, \text{TA} \cdot \sigma}{\Gamma, \text{T}\exists xA \cdot \sigma_{-x}} (T\exists) \qquad \frac{\Gamma, \text{FA} \cdot \sigma}{\Gamma, \text{F}\exists xA \cdot \sigma_{-x}} (F\exists)
\end{array}$$

Figure 17: Calculus \mathbf{I}_{inv}^ξ

(TV) and (C) in C_{inv}^ξ is equivalent to an inference by TV_2 in I_{inv}^ξ . In other words, the last sequents of derivations

$$\frac{\frac{\Gamma, TA \cdot \sigma_1, FC \cdot \delta_1 \quad \Delta, TB \cdot \sigma_2, FC \cdot \delta_2}{\Gamma\theta, \Delta\theta, TA \vee B \cdot \sigma_1\theta, FC \cdot \delta_1\theta, FC \cdot \delta_2\theta} (TV)}{\Gamma\theta\tau, \Delta\theta\tau, TA \vee B \cdot \sigma_1\theta\tau, FC \cdot \delta_1\theta\tau} (C)$$

and

$$\frac{\Gamma, TA \cdot \sigma_1, FC \cdot \delta_1 \quad \Delta, TB \cdot \sigma_2, FC \cdot \delta_2}{\Gamma\vartheta, \Delta\vartheta, TA \vee B \cdot \sigma_1\vartheta, FC \cdot \delta_1\vartheta} (TV_2)$$

are the same. Using the standard arguments of unification theory³ and the fact that for every variable z , if $z \notin (dom(\delta_1) \cup dom(\delta_2))$ then $z\delta_1\theta = z\delta_2\theta$, we prove that $\vartheta = \theta\tau$. \square

This lemma and the arguments used to prove the completeness of C_{inv}^ξ yield the completeness theorem for I_{inv}^ξ :

4.3. THEOREM (Completeness of I_{inv}^ξ). *Let ξ be a closed signed formula. Then ξ is unsatisfiable in intuitionistic logic if and only if ξ has a refutation in I_{inv}^ξ .* \square

Consider an example refutation by the inverse method of the sequent

$$TA \vee B, T\forall x(A \supset P(x, b)), T\forall x(B \supset P(a, x)), F\exists x\exists yP(x, y). \quad (4.2)$$

We show the I_{inv}^ξ -derivation of this sequent in Figure 18. This example demonstrates in particular the necessity of the special form for disjunction rule (TV_2) in I_{inv}^ξ requiring simultaneous unification of two pairs of substitutions⁴.

We would like to note that the resulting inverse method calculus varies depending on the initial tableau calculus. Had we started with the original calculus of Gentzen [1934], we would have obtained a different version of I_{inv}^ξ . An essentially different calculus can be obtained from the multi-succedent sequent formalization of intuitionistic logic [Maehara 1954, Beth 1956], see also [Curry 1963, Troelstra and Schwichtenberg 1996], in which multiple signed formulas FA are allowed in sequents, but some rules are different. An interesting line of research is to obtain “more efficient” (from the viewpoint of the inverse method) calculi. The criteria for efficiency can be the length of propositional derivations, the smaller number of non-invertible rules, the smaller number of pairs of non-permutable rules etc. For tableau-based methods, this line of research is pursued in several publications,

³See e.g., [Apt 1990] where an equivalent notion of unification of a set of equations $\{s_1 = t_1, \dots, s_n = t_n\}$ is considered instead of unification of tuples $\langle s_1, \dots, s_n \rangle$ and $\langle t_1, \dots, t_n \rangle$.

⁴[Tammet 1996, Tammet 1997] included a simplified version of the disjunction rule in his intuitionistic calculi for the inverse method. Essentially, he overlooked the necessity of unifying *two pairs* of substitutions simultaneously.

$$\begin{array}{c}
\frac{\frac{\frac{\mathsf{T}A, \mathsf{F}A \quad \mathsf{T}P(x, b) \cdot \{x \mapsto x_1\}, \mathsf{F}P(x, y) \cdot \{x \mapsto x_1, y \mapsto b\}}{\mathsf{T}A, \mathsf{T}A \supset P(x, b) \cdot \{x \mapsto x_1\}, \mathsf{F}P(x, y) \cdot \{x \mapsto x_1, y \mapsto b\}} \quad (\mathsf{T} \supset)}{\mathsf{T}A, \mathsf{T}Px(A \supset P(x, b)), \mathsf{F}P(x, y) \cdot \{x \mapsto x_1, y \mapsto b\}} \quad (\mathsf{T}V)} \\
\frac{\mathsf{T}A \vee B, \mathsf{T}Px(A \supset P(x, b)), \mathsf{T}Px(B \supset P(a, x)), \mathsf{F}P(x, y) \cdot \{x \mapsto a, y \mapsto b\}}{\mathsf{T}A \vee B, \mathsf{T}Px(A \supset P(x, b)), \mathsf{T}Px(B \supset P(a, x)), \mathsf{F}P(x, y) \cdot \{x \mapsto a, y \mapsto b\}} \quad (\mathsf{F} \exists)} \\
\frac{\mathsf{T}A \vee B, \mathsf{T}Px(A \supset P(x, b)), \mathsf{T}Px(B \supset P(a, x)), \mathsf{F}P(x, y) \cdot \{x \mapsto a, y \mapsto b\}}{\mathsf{T}A \vee B, \mathsf{T}Px(A \supset P(x, b)), \mathsf{T}Px(B \supset P(a, x)), \mathsf{F} \exists x \exists y P(x, y)} \quad (\mathsf{F} \exists)
\end{array}$$

$$\begin{array}{c}
\frac{\mathsf{T}B, \mathsf{F}B \quad \mathsf{T}P(a, x) \cdot \{x \mapsto x_2\}, \mathsf{F}P(x, y) \cdot \{x \mapsto a, y \mapsto x_2\}}{\mathsf{T}B, \mathsf{T}(B \supset P(a, x)) \cdot \{x \mapsto x_2\}, \mathsf{F}P(x, y) \cdot \{x \mapsto a, y \mapsto x_2\}} \quad (\mathsf{T} \supset)} \\
\frac{\mathsf{T}B, \mathsf{T}Px(B \supset P(a, x)), \mathsf{F}P(x, y) \cdot \{x \mapsto a, y \mapsto x_2\}}{\mathsf{T}B, \mathsf{T}Px(B \supset P(a, x)), \mathsf{F}P(x, y) \cdot \{x \mapsto a, y \mapsto x_2\}} \quad (\mathsf{T}V)} \\
\frac{\mathsf{T}B, \mathsf{T}Px(B \supset P(a, x)), \mathsf{F}P(x, y) \cdot \{x \mapsto a, y \mapsto x_2\}}{\mathsf{T}B, \mathsf{T}Px(B \supset P(a, x)), \mathsf{F} \exists x \exists y P(x, y)} \quad (\mathsf{T} \vee_2)
\end{array}$$

Figure 18: A $\mathsf{I}_{\text{inv}}^{\mathsf{f}}$ -derivation

including [Dyckhoff 1992, Shankar 1992, Hudelmaier 1993], but the technique developed for tableau-based methods does not necessarily apply to the inverse method. For example, the calculus of Hudelmaier [1993] is extremely efficient for the tableau-based proof-search in intuitionistic propositional logic, but does not have the subformula property.

4.2. Modal logics

In this section, we cook the inverse method calculi for the propositional versions of several modal logics. As in the previous sections, we develop no semantics for these logics, instead we exploit proof-theoretic properties of cut-free sequent calculi for these logics. For standard formalizations and semantics of modal logics, see e.g. [Kripke 1963, Chagrov and Zakharyashev 1996, Fitting and Mendelsohn 1998] and [Waeler 2001] (Chapter 22 of this Handbook). Proof-theoretic investigations of these logics can be found in [Fitting 1983]. The predicate versions of these logics are described in e.g. [Bowen 1979, Fitting 1983, Fitting and Mendelsohn 1998, Fitting, Thalmann and Voronkov 2000]. We discuss in some detail the design of the inverse method calculi for **K**, for other modal logics we simply give a table of inference rules, leaving the (trivial) proofs to the reader.

Formulas of modal logics have two additional unary connectives \Box and \Diamond . For simplicity, we will only consider formulas in negation normal form, i.e. those built from literals using \wedge , \vee , \forall , \exists , \Box and \Diamond . For a multiset of formulas Γ we denote by $\Box\Gamma$ the multiset of formulas consisting of formulas $\Box A$ for all $A \in \Gamma$. In order to obtain the negation normal form of a modal formula, we use the transformation rules given on page 204 plus the following rules:

$$\begin{aligned}\neg\Box A &\Rightarrow \Diamond\neg A, \\ \neg\Diamond A &\Rightarrow \Box\neg A.\end{aligned}$$

As in the case of negation normal forms for classical logic, the sequents are multisets of formulas (not signed formulas).

4.2.1. Modal logic **K**

The sequent calculus \mathbf{K}_{tab} is obtained from the sequent calculus \mathbf{C}_{tab} for NNFs (Figure 10 on page 205) by adding one inference rule

$$\frac{\Delta, A}{\Gamma, \Box\Delta, \Diamond A} (\Diamond). \quad (4.3)$$

An example refutation in \mathbf{K}_{tab} of a propositional modal formula is shown below.

$$\begin{array}{c} \frac{P, \neg P, \neg Q \quad P, Q, \neg Q}{P, \neg P \vee Q, \neg Q} (\vee) \\ \frac{P, \neg P \vee Q, \neg Q}{\Box P, \Box(\neg P \vee Q), \Diamond\neg Q, \Diamond Q} (\Diamond) \\ \frac{\Box P, \Box(\neg P \vee Q), \Diamond\neg Q, \Diamond Q}{\Box P \wedge \Box(\neg P \vee Q), \Diamond\neg Q, \Diamond Q} (\wedge) \\ \frac{\Box P \wedge \Box(\neg P \vee Q), \Diamond\neg Q, \Diamond Q}{\Box P \wedge \Box(\neg P \vee Q) \wedge \Diamond\neg Q, \Diamond Q} (\wedge) \\ \frac{\Box P \wedge \Box(\neg P \vee Q) \wedge \Diamond\neg Q, \Diamond Q}{\Box P \wedge \Box(\neg P \vee Q) \wedge \Diamond\neg Q \wedge \Diamond Q} (\wedge). \end{array} \quad (4.4)$$

As we can see, this logic also has the subformula property.

We follow the universal recipe of automated deduction. At our disposal are the initial ingredients: the rules of \mathbf{K}_{tab} . Our next aim is to design a suitable calculus dealing with closed formulas (the ground version).

Since we are now much more experienced with cooking, we know an easy way to cook: think of the Subsequent Lemma. In order to find a calculus suitable for the inverse method, we have to extend \mathbf{C}_{inv}^G by rules corresponding to (\diamond) so that the Subsequent Lemma remains true. We will illustrate how such rules can be obtained in an essentially mechanical way by trying to adapt the techniques used in the proofs of this key lemma.

For the Subsequent Lemma to remain true we have to find an inference rule (or a collection of inference rules) with the following two properties: first, the rule should be sound for \mathbf{K} and second, given a derivation of a subsequent of Δ, A we should be able to derive a subsequent of $\Gamma, \Box\Delta, \Diamond A$. Let us try the most obvious candidate:

$$\frac{\Delta, A}{\Box\Delta, \Diamond A} (\diamond). \quad (4.5)$$

Obviously, this rule is sound because it is a special case of (4.3) with $\Gamma = \emptyset$.

Let us verify the second property. Suppose that a subsequent of Δ, A is derivable. Consider two cases.

1. This subsequent contains A , then it has the form Δ', A for $\Delta' \subseteq \Delta$. We can apply an instance of rule (4.5):

$$\frac{\Delta', A}{\Box\Delta', \Diamond A} (\diamond)$$

to obtain a subsequent of $\Gamma, \Box\Delta, \Diamond A$.

2. The subsequent of Δ, A does not contain A . In this case, it has the form Δ' for $\Delta' \subseteq \Delta$. It seems that in general there is no way to derive a subsequent of $\Gamma, \Box\Delta, \Diamond A$ from Δ' only. Therefore, we add one more rule to the inverse method calculus:

$$\frac{\Delta}{\Box\Delta, \Diamond A} (\diamond^+). \quad (4.6)$$

We leave the proof of soundness of this rule to the reader. With the new rule, the Subsequent Lemma holds, since now we can use the inference

$$\frac{\Delta'}{\Box\Delta', \Diamond A} (\diamond^+)$$

to obtain a subsequent of $\Gamma, \Box\Delta, \Diamond A$.

The new rules create no troubles for the Instance Lemma.

By adding to \mathbf{C}_{inv} rules (\diamond) and (\diamond^+) we obtain the *calculus* \mathbf{K}_{inv} .

Our next aim is to add unification to \mathbf{K}_{inv} and cook this calculus into a free-variable calculus. With our greater experience in cooking, this is almost mechanical. Let G be a formula. By adding to \mathbf{C}_{inv}^G the free-variable versions of these rules

$$\frac{\Delta, A \cdot \sigma}{\Box \Delta, \Diamond A \cdot \sigma} (\Diamond) \quad \text{and} \quad \frac{\Delta}{\Box \Delta, \Diamond A \cdot \varepsilon} (\Diamond^+)$$

we obtain the *calculus* K_{inv}^G . The proof of the Instance Lemma for K_{inv}^G is also straightforward.

Below we give an example derivation in K_{inv} which can be obtained by applying the transformation from the Subsequent Lemma to refutation (4.4)

$$\frac{\frac{\frac{P, \neg P \quad Q, \neg Q}{P, \neg P \vee Q, \neg Q} (\vee)}{\Box P, \Box(\neg P \vee Q), \Diamond \neg Q} (\Diamond)}{\Box P, \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q, \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q} (\wedge)$$

$$\frac{\Box P, \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q, \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q}{\Box P, \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q} (C)$$

$$\frac{\Box P \wedge \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q, \Box P \wedge \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q}{\Box P \wedge \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q} (\wedge)$$

$$\frac{\Box P \wedge \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q}{\Box P \wedge \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q} (C)$$

4.3. Other modal logics

In this section we give the inverse method calculi for propositional modal logics **T**, **D**, **S4**, **K4**, and **D4**. Cut-free calculi for these logics used here are taken from [Fitting 1983], but presented in a different notation. Instead of giving a detailed presentation, we simply give a table of inference rules for these logics, both in the tableau form and the inverse form. These rules are given in Figure 19.

The reader can check the subsequent lemma for these calculi. Note that the second (\Diamond) rules of the inverse calculi for **D** and **D4** obtained by a direct translation of the (\Diamond) rules of the tableau calculi, are redundant. Indeed, they can be replaced by the respective (\Box) rules.

5. Naming and connections with resolution

Formulas are too complex objects to be manipulated. During proof-search by the inverse method, sequents tend to grow and become difficult to manage. In this section we consider the *naming technique* that allows one to have a convenient representation of formulas by atoms. The naming technique was already used by Maslov [1968]. The naming technique is also used in other methods of theorem proving in nonclassical logics (e.g., [Horrocks and Patel-Schneider 1999, de Nivelle, Schmidt and Hustadt 2000]), though usually it is not formalized as a logical calculus. Voronkov [2001] introduces a so-called *path calculus* where paths are used instead of the names. The use of the paths considerably simplifies the completeness proof for some redundancies. We consider path calculi in Section 7 of this chapter.

logic	tableau rules	inverse rules
T	$\frac{\Gamma, A}{\Box\Gamma, \Diamond A, \Delta} (\Diamond)$ $\frac{\Gamma, A}{\Gamma, \Box A} (\Box)$	$\frac{\Gamma, A}{\Box\Gamma, \Diamond A} (\Diamond) \quad \frac{\Gamma}{\Box\Gamma, \Diamond A} (\Diamond)$ $\frac{\Gamma, A}{\Gamma, \Box A} (\Box)$
D	$\frac{\Gamma, A}{\Box\Gamma, \Diamond A, \Delta} (\Diamond)$ $\frac{\Gamma}{\Box\Gamma, \Delta} (\Box)$	$\frac{\Gamma, A}{\Box\Gamma, \Diamond A} (\Diamond) \quad \frac{\Gamma}{\Box\Gamma, \Diamond A} (\Diamond)$ $\frac{\Gamma}{\Box\Gamma} (\Box)$
S4	$\frac{\Box\Gamma, A}{\Box\Gamma, \Diamond A, \Delta} (\Diamond)$ $\frac{\Gamma, A}{\Gamma, \Box A} (\Box)$	$\frac{\Box\Gamma, A}{\Box\Gamma, \Diamond A} (\Diamond)$ $\frac{\Gamma, A}{\Gamma, \Box A} (\Box)$
D4	$\frac{\Gamma, \Box\Gamma, A}{\Box\Gamma, \Diamond A, \Delta} (\Diamond)$ $\frac{\Gamma, \Box\Gamma}{\Box\Gamma, \Delta} (\Box)$	$\frac{\Gamma_1, \Box\Gamma_2, A}{\Box\Gamma_1, \Box\Gamma_2, \Diamond A} (\Diamond) \quad \frac{\Gamma_1, \Box\Gamma_2}{\Box\Gamma_1, \Box\Gamma_2, \Diamond A} (\Diamond)$ $\frac{\Gamma_1, \Box\Gamma_2}{\Box\Gamma_1, \Box\Gamma_2} (\Box)$
K4	$\frac{\Gamma, \Box\Gamma, A}{\Box\Gamma, \Diamond A, \Delta} (\Diamond)$	$\frac{\Gamma_1, \Box\Gamma_2, A}{\Box\Gamma_1, \Box\Gamma_2, \Diamond A} (\Diamond) \quad \frac{\Gamma_1, \Box\Gamma_2}{\Box\Gamma_1, \Box\Gamma_2, \Diamond A} (\Diamond)$

The second (\Diamond) rules of the inverse calculi for **D** and **D4** are redundant.

Figure 19: Inference rules for several modal logics

$\exists y \forall x \forall z (P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy)))$	G	$\exists y A(y)$
$\forall x \forall z (P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy)))$	$A(y)$	$\forall x B(x, y)$
$\forall z (P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy)))$	$B(x, y)$	$\forall z C(z, x, y)$
$P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy))$	$C(z, x, y)$	$D(z, x) \wedge E(z, x, y)$
$P(z, x)$	$D(z, x)$	
$\neg P(z, fx) \vee \neg P(x, ffy)$	$E(z, x, y)$	$H(z, x) \vee I(x, y)$
$\neg P(z, fx)$	$H(z, x)$	
$\neg P(x, ffy)$	$I(x, y)$	

Table 1: Subformulas and names

5.1. Naming

In this section we will restrict ourselves to formulas in negation normal form and consider formulas instead of signed formulas. We will try to avoid complete formalization, presenting the technique by examples.

The essence of the naming technique is the following: with each free subformula A of the goal formula G , we associate a unique predicate symbol N , and use these predicate symbols to name arbitrary subformulas in the following way. Let $A(x_1, \dots, x_n)$ be a free subformula of G , where x_1, \dots, x_n are all free variables of this subformula and N be the predicate symbol associated with $A(x_1, \dots, x_n)$, then any subformula $A(t_1, \dots, t_n)$ of G will be represented by the atomic formula $N(t_1, \dots, t_n)$. We say that the atomic formula $N(t_1, \dots, t_n)$ is a **name** of the subformula $A(t_1, \dots, t_n)$.

Since every inference rule of \mathbf{C}_{inv}^G is defined by its principal formula, that is a subformula of G , we also obtain a convenient representation of inference rules. To illustrate the technique, we consider formula (3.7) in negation normal form. The free subformulas of this formula are shown in the left-hand column of Table 1.

We associate with these free subformulas new predicate symbols A, \dots, H , then the names of these subformulas are shown in the middle column of Table 1. In the right-hand column of the table we show the representation of the subformulas of G through the names of its immediate subformulas.

Formally, let G be the goal formula, N_1, N_2, \dots be an alphabet of predicate symbols not occurring in G and x_1, x_2, \dots be all variables of G . Let all free subformulas of G be A_1, \dots, A_n . Let $\bar{y}_1, \dots, \bar{y}_n$ be the sequences of variables of A_1, \dots, A_n , respectively, the variables in each sequence \bar{y}_i are in the order of their first occurrences in A_i . We call the atomic formulas $N_i(\bar{y}_i)$ the **names** of the free subformulas $A_i(\bar{y}_i)$ of G . Likewise, let $A_i(\bar{t})$ be any subformula of G . We call the atomic formula $N_i(\bar{t})$ the **name** of $A_i(\bar{t})$.

For example, in Table 1 the atomic formula $E(z, x, y)$ is the name of the free subformula $\neg P(z, fx) \vee \neg P(x, ffy)$ of G . Then for every term t_z, t_x, t_y , the atomic

formula $E(t_z, t_x, t_y)$ is the name of the subformula $\neg P(t_z, ft_x) \vee \neg P(t_x, fft_y)$ of G .

We will now modify the calculus C_{inv}^G by replacing subformulas by their names. We call the resulting calculus the **name calculus for G** . Every non-literal free subformula A of G induces one or more inference rules of C_{inv}^G with A as the principal formula. Consider, for example, the free subformula $\neg P(z, fx) \vee \neg P(x, ffy)$. The calculus C_{inv}^G contains the following inference rule

$$\frac{\Gamma, \neg P(z, fx) \cdot \sigma_1 \quad \Delta, \neg P(x, ffy) \cdot \sigma_2}{\Gamma\theta, \Delta\theta, \neg P(z, fx) \vee \neg P(x, ffy) \cdot \sigma_1\theta} (\vee). \quad (5.1)$$

Note that σ_1 has the form $\{z \mapsto s_z, x \mapsto s_x\}$, for some terms s_z, s_x . Likewise, σ_2 has the form $\{x \mapsto t_x, y \mapsto t_y\}$, for some terms t_x, t_y . It is not hard to argue that θ has the form $\{z \mapsto s_z, y \mapsto t_y\}\tau$, where τ is a most general unifier of s_x and t_x . Therefore, the rule can be rewritten as

$$\frac{\Gamma, \neg P(s_z, fs_x) \quad \Delta, \neg P(t_x, fft_y)}{(\Gamma, \Delta, (\neg P(s_z, fs_x) \vee \neg P(s_x, fft_y))\tau} (\vee), \quad (5.2)$$

where τ is a most general unifier of s_x, t_x . By replacing the principal and side formulas of this rule by their names, we obtain the following inference rule of the name calculus for G :

$$\frac{\Gamma, H(s_z, s_x) \quad \Delta, I(t_x, t_y)}{(\Gamma, \Delta, E(s_z, s_x, t_y))\tau} (\vee), \quad (5.3)$$

where τ is a most general unifier of s_x and t_x .

Consider another example. Since the goal formula G contains occurrences of two literals $P(z, x), \neg P(z, fx)$, the calculus C_{inv}^G contains the axiom

$$P(z, x) \cdot \{x \mapsto fx_2, z \mapsto x_1\}, \neg P(z, fx) \cdot \{x \mapsto x_2, z \mapsto x_1\}.$$

By replacing the two literals by their names and applying the substitutions, we obtain the following axiom of the name calculus for G :

$$D(x_1, fx_2), H(x_1, x_2).$$

In total, the name calculus for $\exists y \forall x \forall z (P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy)))$ contains the axioms and inference rules shown in Figure 20 (compare them with the subformulas and names of Table 1):

An example derivation in the name calculus is given in Figure 21. This derivation can be obtained from the derivation of Figure 14 by replacing subformulas by their names. Note that the derivation in the name calculus is much more compact than the original derivation in C_{inv}^G .

5.2. The inverse method as resolution

Sequents in name calculi are multisets of atoms, i.e., they can be considered as (positive) clauses in resolution. In this paper a **clause** is a multiset of literals.

$$\begin{array}{c}
\frac{\Gamma, A(s_y)}{\Gamma, G} (\exists) \qquad \frac{\Gamma, B(s_x, s_y)}{\Gamma, A(s_y)} (\forall_1) \qquad \frac{\Gamma, C(s_x, s_x, s_y)}{\Gamma, B(s_x, s_y)} (\forall_2) \\
\\
\frac{\Gamma, D(s_x, s_x)}{\Gamma, C(s_x, s_x, y)} (\wedge_l) \qquad \frac{\Gamma, E(s_x, s_x, s_y)}{\Gamma, C(s_x, s_x, s_y)} (\wedge_r) \\
\\
\frac{\Gamma, H(s_x, s_x) \quad \Delta, I(t_x, t_y)}{(\Gamma, \Delta, E(s_x, s_x, t_y))\text{mgu}(s_x, t_x)} (\vee) \\
\\
\frac{}{D(x_1, fx_2), H(x_1, x_2).} (Ax_1) \qquad \frac{}{D(x_1, ffx_2), I(x_1, x_2).} (Ax_2) \\
\\
\frac{\Gamma, F_1, F_2}{(\Gamma, F_1)\sigma} (C)
\end{array}$$

The rule (\exists) satisfies the eigenvariable condition: s_y is a variable not occurring in Γ . In the contraction rule (C) σ a most general unifier of F_1 and F_2 .

Figure 20: The name calculus for $\exists y \forall x \forall z (P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffy)))$

- | | | |
|------|---|------------------------|
| (1) | $D(x_1, fx_2), H(x_1, x_2)$ | (Ax_1) |
| (2) | $D(y_1, ffy_2), I(y_1, y_2)$ | (Ax_2) |
| (3) | $D(x_1, fx_2), D(x_2, ffx_3), E(x_1, x_2, x_3)$ | (\vee) from (1,2) |
| (4) | $D(fx_1, ffx_1), E(fx_1, fx_1, x_1)$ | (C) from (3) |
| (5) | $D(fx_1, ffx_1), C(fx_1, fx_1, x_1)$ | (\wedge_r) from (4) |
| (6) | $D(fx_1, ffx_1), B(fx_1, x_1)$ | (\forall_2) from (5) |
| (7) | $D(fx_1, ffx_1), A(x_1)$ | (\forall_1) from (6) |
| (8) | $C(fx_1, ffx_1, x_2), A(x_1)$ | (\wedge_l) from (7) |
| (9) | $B(ffx_1, x_2), A(x_1)$ | (\forall_2) from (8) |
| (10) | $A(x_2), A(x_1)$ | (\forall_1) from (9) |
| (11) | $A(x_1)$ | (C) from (10) |
| (12) | G | (\exists) from (11) |

Figure 21: Derivation in the name calculus corresponding to the derivation of Figure 14.

There exists a close relationship between inferences in the name calculus and some inferences by hyperresolution. Consider, for example, inference rule (5.3) of the name calculus. One possible inference by this rule (taken from the derivation of Figure 21) is

$$\frac{D(x_1, fx_2), H(x_1, x_2) \quad D(x_4, ffx_3), I(x_4, x_3)}{D(x_1, fx_2), D(x_2, ffx_3), E(x_1, x_2, x_3)} (\vee).$$

Note that exactly the same conclusion can be obtained by the following application of the positive hyperresolution rule⁵ with the additional clause $\neg H(z, x), \neg I(x, y), E(z, x, y)$.

$$\frac{D(x_1, fx_2), H(x_1, x_2) \quad D(x_4, ffx_3), I(x_4, x_3) \quad \neg H(z, x), \neg I(x, y), E(z, x, y)}{D(x_1, fx_2), D(x_2, ffx_3), E(x_1, x_2, x_3)} (\text{hyper}).$$

This observation can be generalized: every inference by (5.3) in the name calculus can be made into an inference by hyperresolution by adding the additional premise $\neg H(z, x), \neg I(x, y), E(z, x, y)$. Every such inference by hyperresolution with one of the premises $\neg H(z, x), \neg I(x, y), E(z, x, y)$ can be made into an inference by (5.3) by removing this premise.

Likewise, any rule (\vee) of the name calculus for any formula G can be simulated by hyperresolution. Moreover, this is true for every rule (\wedge_l), (\wedge_r) and (\forall). The only rule that cannot be directly simulated by hyperresolution is (\exists), because of the eigenvariable condition. However, the quantifier \exists never occurs in Skolemized formulas, so for Skolemized formulas the inverse method can be simulated by hyperresolution.

Formally, the transformation works as follows. Let G be a rectified formula in negation normal form without occurrences of \exists . We will now build two sets of clauses: a set Ax_G of positive clauses and a set $Rules_G$ of non-positive clauses. **The set Ax_G** is exactly the set of axioms of the name calculus for G and can be described as follows.

- Suppose that A, B is a pair of atomic formulas such that A and $\neg B$ are free subformulas of G , and A and B are weakly unifiable. Then Ax_G contains a clause $A'\rho\tau, B'\tau$ such that A', B' are the names of $A, \neg B$ respectively, ρ renames A away from B , and τ is a most general unifier of $A\rho$ and B .

The **set $Rules_G$** is described as follows.

- Suppose that $A \vee B$ is a free subformula of G , and the atoms A', B', C' are the names of $A, B, A \vee B$, respectively. Then $Rules_G$ contains the clause $\neg A', \neg B', C'$.
- Suppose that $A \wedge B$ is a free subformula of G , and the atoms A', B', C' are the names of $A, B, A \wedge B$, respectively. Then $Rules_G$ contains the clauses $\neg A', C'$ and $\neg B', C'$.

⁵Hyperresolution is discussed in [Bachmair and Ganzinger 2001, page 61] (Chapter 2 of this Handbook).

- Suppose that $\forall xA$ is a free subformula of G , and the atoms A', B' are the names of $A, \forall xA$, respectively. Then $Rules_G$ contains the clause $\neg A', B'$.

5.1. LEMMA (Bisimulation of Inferences). (i) *For every inference different from contraction*

$$\frac{A_1 \quad \cdots \quad A_n}{A}$$

in the name calculus for G , there exists a clause $C \in Rules_G$ such that

$$\frac{A_1 \quad \cdots \quad A_n \quad C}{A}$$

is a inference by positive hyperresolution.

(ii) *For every clause $C \in Rules_G$ and every inference by positive hyperresolution*

$$\frac{A_1 \quad \cdots \quad A_n \quad C}{A}$$

the figure

$$\frac{A_1 \quad \cdots \quad A_n}{A}$$

is a valid inference in the name calculus for G .

We leave the (straightforward) proof of this lemma to the reader.

Since the axioms in the name calculus for G are precisely the clauses in Ax_G , Lemma 5.1 implies the following lemma.

5.2. LEMMA (Bisimulation of Derivations). (i) *Every derivation \mathcal{N} in the name calculus for G can be made into a positive hyperresolution derivation from the set of clauses $Ax_G \cup Rules_G$ by adding a clause from $Rules_G$ to the premises of inferences of \mathcal{N} different from (Ax) or (C) .*

(ii) *For every derivation \mathcal{D} by positive hyperresolution from the set of clauses $Ax_G \cup Rules_G$ the figure obtained from \mathcal{D} by removing all clauses in $Rules_G$ is a valid derivation in the name calculus for G .*

5.3. EXAMPLE. Let us illustrate this theorem. Consider the Skolemized form of formula (3.7) used in Example 3.21 on page 206:

$$G = \forall x \forall z (P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffc))).$$

Introduce the following names for the free subformulas of G :

$$\begin{aligned} H &= \forall x \forall z (P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffc))), \\ A(x) &= \forall z (P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffc))), \\ B(x, z) &= P(z, x) \wedge (\neg P(z, fx) \vee \neg P(x, ffc)), \\ C(z, x) &= P(z, x), \\ D(z, x) &= \neg P(z, fx) \vee \neg P(x, ffc), \\ E(z, x) &= \neg P(z, fx), \\ F(x) &= \neg P(x, ffc). \end{aligned}$$

The set of clauses Ax_G consists of two clauses:

- (1) $C(z, fx), E(z, x).$
- (2) $C(z, ffc), F(z).$

The set of clauses $Rules_G$ consists of five clauses:

- (3) $\neg A(x), H.$
- (4) $\neg B(x, z), A(x).$
- (5) $\neg C(z, x), B(x, z).$
- (6) $\neg D(z, x), B(x, z).$
- (7) $\neg E(z, x), \neg F(x), D(z, x).$

A hyperresolution derivation of H from the initial set of clauses (1)–(7) is as follows:

- | | | |
|------|-------------------------------------|---------------------|
| (8) | $B(fx, z), E(z, x)$ | hyper from (1,5). |
| (9) | $B(ffc, z), F(z)$ | hyper from (2,5). |
| (10) | $B(fx, z), B(ffc, x), D(z, x)$ | hyper from (8,9,7). |
| (11) | $B(ffc, fc), B(ffc, fc), D(fc, fc)$ | factor from (10). |
| (12) | $B(ffc, fc), D(fc, fc)$ | factor from (10). |
| (13) | $B(ffc, fc), B(fc, fc)$ | hyper from (12,6). |
| (14) | $A(ffc), B(fc, fc)$ | hyper from (13,4). |
| (15) | $A(ffc), A(fc)$ | hyper from (14,4). |
| (16) | $H, A(fc)$ | hyper from (15,3). |
| (17) | H, H | hyper from (16,3). |
| (18) | H | factor from (17). |

The reader can try to convert this derivation into a Gentzen-style derivation of the original formula, for example in the calculus C_{inv} .

In resolution theorem proving, we are usually interested in finding a *refutation* from a set of clauses, instead of constructing a derivation of a goal. However, when the goal is a ground atom G , it is derivable from a set of clauses S if and only if the set of clauses $S \cup \{\neg G\}$ is inconsistent. Therefore we have

5.4. THEOREM. *Let G be a closed formula in negation normal form without occurrences of \exists and H be the name of G . Then G is inconsistent if and only if so is the set of clauses $Ax_G \cup Rules_G \cup \{\neg H\}$.* \square

The use of the naming technique and the simulation of the inverse method by resolution are interesting for several reasons. One reason is the simple representation of sequents by clauses. Another reason is that some strategies and redundancies known for resolution can be transferred from resolution to the inverse method. This is not so important for theorem proving in classical logic, where we can use the best

implementations of resolution on the transformed set of clauses directly,⁶ but this is invaluable for nonclassical logics. Indeed, as demonstrated in [Voronkov 1992], many strategies complete for resolution for classical logic have analogues complete for nonclassical logics.

5.3. Optimized translation

We will demonstrate how the use of some elementary properties of resolution-based theorem proving can improve the inverse method for classical logic, by formulating a more concise name calculus. In this name calculus only so-called *disjunctive subformulas* have names.

We will use the following simple property of clause form logic.

5.5. LEMMA (Definition Elimination). *Suppose that P is a predicate symbol and S is a set of clauses. Suppose that P has at most one occurrence in each clause in S . Denote by S' the set of clauses obtained from S by adding, for each pair of clauses $P(\bar{s}) \wedge C$ and $\neg P(\bar{t}), D$ such that \bar{s} and \bar{t} are unifiable, the clause $(C, D)\text{mgu}(\bar{s}, \bar{t})$ and by S'' the clause obtained from S' by deleting all clauses in which P occurs. Then S is inconsistent if and only if so is S'' .* \square

We will now show how to apply this lemma and Theorem 5.4 to obtain an optimized translation into a set of clauses. Using this optimized translation we can design a new calculus for Skolemized formulas of classical logic with no (\wedge_l) , (\wedge_r) , or (\forall) rules, so the only remaining rules are (\vee) . We call *disjunctive free subformulas of G* all free subformulas F_1 and F_2 of G such that $F_1 \vee F_2$ is a free subformula of G . Our aim is now to eliminate names for all non-disjunctive free subformulas from the name calculus. This can be done by a straightforward application of the Definition Elimination Lemma.

5.6. EXAMPLE. We will illustrate the optimized translation on the formula of Example 5.3 on page 225. Let us show how to eliminate all names for non-disjunctive subformulas. Before eliminating the names, let us first add to the set of clauses the negation of the name of the goal formula $\neg H$. Note that there are only two names of disjunctive free subformulas, namely $E(z, x)$ and $F(x)$. Elimination of all other names is illustrated in Figure 22. At each step of the elimination procedure we shade the literals with the currently eliminated atom, corresponding to the literals $P(\bar{s})$ and $\neg P(\bar{t})$ of the Definition Elimination Lemma.

It is not hard to argue that after a finite number of steps all names for non-disjunctive free subformulas will be eliminated. Moreover, the size of the resulting

⁶Indeed the currently fastest methods of first-order theorem proving take from the inverse method only the idea of the short clause form translation and then apply resolution strategies to the obtained "short" set of clauses [Weidenbach, Gaede and Rock 1996, Nonnengart and Weidenbach 1998].

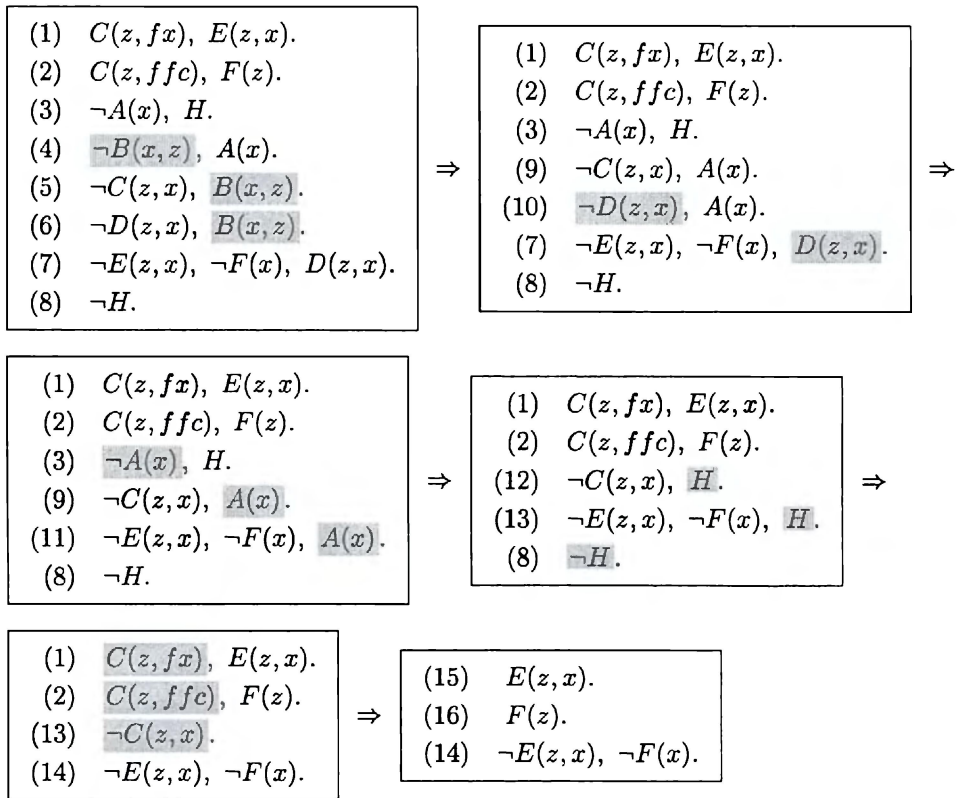


Figure 22: Elimination of names for non-disjunctive subformulas

set of clauses is less than the size of the original set. In fact, we can also eliminate some names for disjunctive subformulas as well, but this may result in the exponential blow-up of the size of the set of clauses.

We have demonstrated how to obtain the optimized translation by using definition elimination from the result of the nonoptimized translation, but the optimized translation can be formulated directly in terms of the goal formula. Essentially, instead of using the name of a subformula, one has to use the name of its *least disjunctive superformula*; see for details [Degtyarev and Voronkov 1994b, Degtyarev and Voronkov 1995a].

The technique that allows one to use only the names for disjunctive subformulas was used in a number of papers [Voronkov 1985, Voronkov 1990b, Degtyarev and Voronkov 1994a, Degtyarev and Voronkov 1995a, Degtyarev and Voronkov 1996g]. Similar technique leading to more concise name calculi for nonclassical logics were described in various forms in [Voronkov 1992, Mints et al. 1996, Voronkov 2001].

5.4. Why hyperresolution

Theorem 5.4 can be proved by simpler techniques as demonstrated in [Boy de la Tour 1990, Degtyarev and Voronkov 1994a]. The transformation of G into clausal form can be performed in polynomial time, unlike the standard transformation that can lead to exponentially large clausal forms. The use of the naming technique together with other techniques for obtaining short clausal forms is discussed in [Nonnengart and Weidenbach 2001] (Chapter 6 of this Handbook).

The use of hyperresolution on the transformed set of clauses is motivated by one observation. Since there is a *bisimulation* between inferences in the name calculus for G and hyperresolution inferences, the hyperresolution derivations can be transformed back to the derivations in the name calculus, which, in turn, have a simple translation into cut-free Gentzen-style derivations. Thus, we can easily transform hyperresolution derivations into Gentzen-style derivations.

Note that the result of a naming transformation is a set of clauses that is unsatisfiable if and only if so is the original formula. In general, it is not necessary to use positive hyperresolution in order to establish inconsistency of a set of clauses. Any other resolution-based methods can be used instead.

5.5. Inference rules not captured by hyperresolution

Italians believe that fish should never be cooked with cheese. This belief is not shared by some other cuisines. What happens when we deal with a calculus which has rules that do not fit into the (hyper)resolution framework, but still want to present them in the form of a resolution calculus? The result is something that is liked by some people and regarded as inappropriate cooking style by others.

A rich source of unusual inference rules is intuitionistic logic (both propositional and predicate). Let us consider several examples.

We will assume that we use a naming technique, so that each free-subformula of the goal signed formula is given a name, that is a predicate symbol, and try to simulate inference rules of the inverse method by special kind of clauses, like the clauses in $Rules_G$ used in the naming transformation for classical logic. We will denote the name of a free signed subformula α by N_α .

Most rules of I_{inv}^ξ of Figure 17 on page 214 can be easily presented as resolution rules. For example, the rule

$$\frac{\Gamma, FA \cdot \sigma_1 \quad \Delta, FB \cdot \sigma_2}{\Gamma\theta, \Delta\theta, FA \wedge B \cdot \sigma_1\theta} (F \wedge)$$

can be simulated by adding the clause

$$\neg N_{FA}(\bar{x}), \neg N_{FB}(\bar{y}), N_{FA \wedge B}(\bar{z}),$$

where $\bar{x}, \bar{y}, \bar{z}$ are the all variables of the respective signed subformulas. Then any inference by $(F \wedge)$ is simulated by two resolution inferences or one hyperresolution inference with this clause. However, the rule

$$\frac{\Gamma, TA \cdot \sigma_1, FC \cdot \delta_1 \quad \Delta, TB \cdot \sigma_2, FC \cdot \delta_2}{\Gamma\vartheta, \Delta\vartheta, TA \vee B \cdot \sigma_1\vartheta, FC \cdot \delta_1\vartheta} (T \vee_2)$$

is more problematic, since standard resolution has no rules that would require unification of two literals in one clause with two literals in another clause. Even propositional intuitionistic logic gives us some surprises. An example is the rule

$$\frac{\Gamma, TA, FA \supset B}{\Gamma\vartheta, FA \supset B} (F \supset_{lr})$$

which again requires to apply a rule based on *two* occurrences of formulas in the same sequent. If we try to simulate this rule by a seemingly appropriate clause

$$\neg N_{TA}, N_{FA \supset B},$$

we will get an unsound calculus.

Other sequent calculus rules that cannot be captured by standard hyperresolution are the rules with the eigenvariable conditions.

Several “resolution” calculi described in the literature, for example [Mints 1990, Tammet 1996], are in fact inverse calculi in which inference rules are presented as special kind of “input clauses”. The inference rules of such calculi are modifications of the standard resolution rule, in which one of the parents is an input clause.

For example, one of the rules in [Mints 1990] is

$$\frac{p, q \Rightarrow r \text{ (input)} ; \quad U \rightarrow p ; \quad V \rightarrow q}{U, V \Rightarrow r}$$

which is essentially the $(F \wedge)$ rule of the inverse calculus.

5.6. Further issues to naming

Names give a very concise representation of subformulas of the goal formula. Since the names are simply atomic formulas, they are easier to handle than arbitrary formulas. There are variations of the naming technique, let us describe two of them here.

5.6.1. No axioms

The rules for forming the set Ax_G corresponding to the axioms of the inverse method calculus can be replaced by less complicated ones, denoted Ax'_G . Instead of building these axioms by unifying every pair of literals, we can replace Ax_G by the set of clauses obtained as follows.

For every free subformula L of G such that L is a literal do the following. Take the name A of L and add the clause \bar{L}, A to Ax'_G .

Consider Example 5.3 on page 225 again. If we use this naming scheme, instead of clauses (1) and (2) of that example we obtain the following three clauses:

- (a) $\neg P(z, x), C(z, x).$
- (b) $P(z, fx), E(z, x).$
- (c) $P(x, ffc), F(x).$

There is no more exact correspondence between the hyperresolution derivations from the transformed set of clauses and inverse method derivations. However, it is easy to see that each member of Ax_G can be constructed in one hyperresolution inference between two clauses in Ax'_G resolving two literals of the original signature. For example, clause (1) is obtained by the following hyperresolution inference from the clauses (a) and (b).

$$\frac{\neg P(z, x), C(z, x) \quad P(z, fx), E(z, x)}{C(z, fx), E(z, x)} \text{ (hyper)}$$

We can obtain a near exact correspondence between the inverse method derivations and ordered hyperresolution derivations for an ordering in which the atoms of the old signature are greater than the new names.

5.6.2. Formula matching

Suppose that the goal G has two free subformulas F_1 and F_2 such that $F_2 = F_1\sigma$ for some substitution σ . Further, let A_1, A_2 be the names of F_1, F_2 respectively. Then one can replace A_2 by $A_1\sigma$ in all clauses. This optimization can be used for either variant of the naming transformation, using Ax_G or Ax'_G .

Consider again Example 5.3 on page 225. One can see that the free subformula $\neg P(x, ffc)$ is an instance of the free subformula $\neg P(z, fx)$ with the substitution $\{z \mapsto x, x \mapsto fc\}$. Therefore, instead of the name $F(x)$ of $P(x, ffc)$, one can use $E(z, x)\{z \mapsto x, x \mapsto fc\} = E(x, fc)$. For the naming transformation with Ax_G , this would result in the replacement of clauses (2) and (7) by

- (2') $C(z, ffc), E(z, fc).$
- (7') $\neg E(z, x), \neg E(x, fc), D(z, x).$

For the naming transformation with Ax'_G , this would result in the replacement of clauses (2) and (c) by (2') and

$$(c') \quad P(x, ffc), E(x, fc).$$

After this the clause (c') becomes an instance of the clause (b) and can be removed.

One can give more complex examples, with non-literal subformulas F_1 and F_2 . In principle, for this translation to be valid it is enough to know that F_2 is *equivalent* to $F_1\sigma$, but the problem of the existence of such a substitution σ is undecidable. Moreover, even for a fixed σ , the equivalence of $F_1\sigma$ and F_2 is undecidable. In

practice, one can use weaker criteria for equivalence, for example equivalence w.r.t. renaming of bound variables, associativity and commutativity of \wedge and \vee .

The naming technique is also important in the development of resolution decision procedures [Fermüller et al. 2001] and in establishing simulation results between various refinements of resolution [Bachmair and Ganzinger 2001] (Chapters 25 and 2 of this Handbook).

5.7. Fast food warning

With your new experience in cooking, you can now try the recipe yourself: take your favorite logic and cook the inverse method calculus for it (or ask your students to do so). With most logics you can do it rather quickly and in the most straightforward way. The aim of this section is to warn you: beware of fast food! And you don't go with your students to fast food restaurants, do you? Your meal is not yet delicious: you forgot to season it to perfection. Be patient and wait until you read about seasoning.

6. Season your meal: strategies and redundancies

Recall that all the inverse method calculi designed so far, for example, \mathbf{K}_{inv}^G , have the *finite rule property*.

1. Every formula occurring in a derivation of a closed formula G is a subformula of G .
2. There exists a finite number of axioms.
3. For every sequent Γ , there exists only a finite number of ways of applying an inference rule to it (for example, (\wedge_l) , (\wedge_r) , (\diamond) , (\diamond^+) , and (C) in the case of propositional \mathbf{K}_{inv}^G).
4. For every pair of sequents Γ and Δ , there exists only a finite number of ways of applying an inference rule to Γ and Δ (for example, (\vee) in the case of propositional \mathbf{K}_{inv}^G).

A typical refutation procedure based on the inverse method uses these properties and works roughly as follows.

1. Let S be a set of sequents, initially $\{p, \neg p \mid p, \neg p \text{ are subformulas of } G\}$.
2. Repeatedly apply all possible inference rules to sequents in S , adding their conclusions to S until no new sequents can be obtained or a refutation of G is found.

However, such naive procedures are hopelessly inefficient because the search space (the set of sequents S) grows rapidly, unless they are augmented with powerful **redundancy criteria**. There are two kinds of redundancy criteria, allowing us to get rid of **redundant sequents** and **redundant inferences**. Intuitively, a sequent or inference is redundant if it can be omitted without losing completeness. Very

powerful techniques for proving redundancies have been developed in resolution-based calculi for classical logic, see e.g., [Bachmair and Ganzinger 2001, Nieuwenhuis and Rubio 2001] (Chapters 2 and 7 of this Handbook).

Redundancy criteria can be incompatible. For example, it is possible that some sequents Γ and Δ are redundant, but every refutation contains at least one of these sequents. Therefore, one usually proves completeness for a collection of redundancy criteria. If we prove several redundancy criteria for a calculus, the proof-search by the inverse method works as follows.

1. Let S be a set of sequents, initially $\{p, \neg p \mid p, \neg p \text{ are subformulas of } G\}$ *without the redundant sequents of this form*.
2. Repeatedly apply all *nonredundant* inferences to sequents in S , adding to S those conclusions of these inferences *that are nonredundant*.
3. Remove all sequents that *become redundant* due to the addition of these conclusions of inferences.

A number of redundancy criteria for various nonclassical logics are proved in [Voronkov 1992] using a rather general technique (though many proofs are absent). There is a paper on the inverse method for **S4** [Mints et al. 1996]. This paper is interesting because it discusses many redundancy criteria, but proofs of many statements are missing. In particular, [Mints et al. 1996] use some strategies *deleting* clauses from the search space, though no general technique for proving completeness of calculi with deletion is developed.

To prove completeness of calculi with redundancies in resolution-based theorem proving for *classical* logic, a rather general model-theoretic technique has been developed. Completeness of nearly all known redundancy criteria can be inferred from very general statements about redundant inferences and clauses. However, such a model-theoretic technique was not developed for nonclassical logics. Two different proof-theoretic techniques for nonclassical logics are presented in [Voronkov 1992] and [Voronkov 2001, Voronkov 2000b]. In the following sections we will explain the technique of [Voronkov 2001, Voronkov 2000b] in detail.

The idea is based on proving properties of refutations in the tableau calculus \mathbf{K}_{tab} and transforming them into redundancy criteria for the inverse method using a suitable Subsequent Lemma. However, the calculus \mathbf{K}_{tab} by itself is not very useful for proving properties of derivations, as many properties are best formulated in terms of occurrences of subformulas in the goal formula. To have a suitable language for speaking about subformulas, we give a reformulation of the sequent calculus as a *path calculus*.

7. Path calculi

In this section we define *path calculi* obtained from the inverse method calculi by using *paths* instead of subformulas. Paths encode the position of subformulas in the goal formula. The use of path is similar to the use of the naming technique, but paths (unlike names) contain a lot of information about the occurrence of a subformula in the goal formula. Most importantly, paths encode sequences of inferences needed

to obtain the goal formula from subformulas. Moreover, as we show in the next section, the notion of paths allows one to formulate redundancies in sequent calculi in a simple way.

In this section we will introduce path calculi only for propositional \mathbf{K} , the generalization to other logics with the subformula property is straightforward. The material of this section is based upon [Voronkov 2000b, Voronkov 2001]. We will first introduce a tableau-style path calculus for \mathbf{K} , and then an inverse method calculus. In this section we assume all formulas are in negation normal form.

7.1. The tableau path calculus

We recall the tableau calculus \mathbf{K}_{tab} for propositional \mathbf{K} presented in Section 4.2.1. It will be helpful to observe the following properties of \mathbf{K}_{tab} .

7.1. LEMMA. (i) \mathbf{K}_{tab} has the **subformula property**: every formula occurring in a derivation of a sequent Γ consists of subformulas of formulas of Γ . (ii) If a sequent Γ has a refutation, then any sequent Γ, Δ has a refutation too. (iii) The rules (\wedge) and (\vee) are invertible.

In this and the next section G will always denote the **goal formula**.

7.2. DEFINITION (path). A **path** is any finite sequence of symbols $\wedge_l, \wedge_r, \vee_l, \vee_r, \Box, \Diamond$. An example of a path is $\wedge_l \Diamond \Diamond \vee_r$. The empty path is denoted ϵ . Let G be a formula. The notion of **path in G** , or **G -path**, and the **subformula of G on a path π** , denoted $G|_\pi$, are defined inductively as follows.

1. The empty path ϵ is a G -path, and $G|_\epsilon$ is G .
2. Suppose that π is a G -path and $G|_\pi = F$. Then
 - (a) If F has the form $F_1 \wedge F_2$, then $\pi \wedge_l$ and $\pi \wedge_r$ are G -paths, and $G|_{\pi \wedge_l} = F_1$, $G|_{\pi \wedge_r} = F_2$. In this case we call π a **\wedge -path**.
 - (b) If F has the form $F_1 \vee F_2$, then $\pi \vee_l$ and $\pi \vee_r$ are G -paths, and $G|_{\pi \vee_l} = F_1$, $G|_{\pi \vee_r} = F_2$. In this case we call π a **\vee -path**.
 - (c) If F has the form $\Box F_1$, then $\pi \Box$ is a G -path and $G|_{\pi \Box} = F_1$. In this case we call π a **\Box -path**.
 - (d) If F has the form $\Diamond F_1$, then $\pi \Diamond$ is a G -path and $G|_{\pi \Diamond} = F_1$. In this case we call π a **\Diamond -path**.

A path π is a **literal path** if $G|_\pi$ is a literal.

Note that the notions like \wedge -path or literal path make sense only when we speak about G -paths for a fixed formula G , and make no sense for paths in general.

We will use the formula

$$E = \Box P \wedge \Box(\neg P \vee Q) \wedge \Diamond \neg Q \wedge \Diamond Q \quad (7.1)$$

for nearly all examples of this section. We consider conjunction and disjunction right-associative, so, for example, formula (7.1) denotes $\Box P \wedge (\Box(\neg P \vee Q) \wedge (\Diamond \neg Q \wedge \Diamond Q))$. The notion of subformula on a path is illustrated in Figure 23.

Evidently, for each subformula F of G , there is a G -path π such that $G|_{\pi} = F$. The tableau calculus \mathbf{K}_{tab} used for refuting a formula G can be turned into a calculus where the subformulas of G are replaced by the corresponding paths. We call a **path sequent** any finite multiset of paths. When no ambiguity arises, we will refer to path sequents simply as sequents. For a path sequent $\Pi = \pi_1, \dots, \pi_n$ we denote by $\Pi \Box$ the path sequent $\pi_1 \Box, \dots, \pi_n \Box$. The **path calculus** \mathbb{P}_{tab}^G and a refutation in it are given in Figures 24 and 25.

The following **bisimulation lemma** is the key to proving completeness of the path calculus and various refinements thereof. Let us call the **formula image** of a path sequent or a derivation in \mathbb{P}_{tab}^G the figure obtained from this path sequent or derivation by replacing every path π by the formula $G|_{\pi}$.

7.3. LEMMA (Bisimulation Lemma for \mathbb{P}_{tab}^G). (i) Let D be a derivation in \mathbb{P}_{tab}^G . Then the formula image of D is a derivation of G in \mathbf{K}_{tab} . (ii) Let D' be a derivation of a sequent A_1, \dots, A_n in \mathbf{K}_{tab} , and π_1, \dots, π_n be positions such that $G|_{\pi_i} = A_i$, for all $i = 1, \dots, n$. Then there exists a derivation D of π_1, \dots, π_n in \mathbb{P}_{tab}^G such that D' is the formula image of D . (iii) Both (i) and (ii) also hold if “derivation” is replaced by “refutation” everywhere.

PROOF. (i) is proved by the straightforward analysis of inferences in \mathbb{P}_{tab}^G . (iii) easily follows from (i) and (ii). We will only show how to prove (ii). The proof is by induction on the number of inferences in D' . We consider only one case, when the last inference is (\vee), other cases are similar. In this case without loss of generality we assume that $A_n = B \vee C$ and that A_n is the principal formula of the last inference. Then the derivation D' has the form

$$\frac{\begin{array}{c} \vdots D'_1 \\ A_1, \dots, A_{n-1}, B \end{array} \quad \begin{array}{c} \vdots D'_2 \\ A_1, \dots, A_{n-1}, C \end{array}}{A_1, \dots, A_{n-1}, B \vee C} (\vee).$$

Evidently, $\pi_n \vee_l$ and $\pi_n \vee_r$ are paths in G . Applying the induction hypothesis to the derivations D'_1 and D'_2 , we obtain derivations D_1 and D_2 such that D'_1 and D'_2 are their formula images. Define D as the derivation

$$\frac{\begin{array}{c} \vdots D_1 \\ \pi_1, \dots, \pi_{n-1}, \pi_n \vee_l \end{array} \quad \begin{array}{c} \vdots D_2 \\ \pi_1, \dots, \pi_{n-1}, \pi_n \vee_r \end{array}}{\pi_1, \dots, \pi_{n-1}, \pi_n} (\vee).$$

It is not hard to argue that D' is the formula image of D . □

As a corollary of the Bisimulation Lemma and completeness of \mathbf{K}_{tab} , we obtain

7.4. THEOREM (Completeness of \mathbb{P}_{tab}^G). A formula G is unsatisfiable if and only if ϵ has a refutation in \mathbb{P}_{tab}^G .

PROOF. By the completeness of \mathbf{K}_{tab} , unsatisfiability of G is equivalent to the existence of a refutation of G in \mathbf{K}_{tab} .

\Rightarrow Let ϵ have a refutation \mathbb{P}_{tab}^G . By the Bisimulation Lemma there exists a refutation in \mathbf{K}_{tab} of the formula $G|_\epsilon$, i.e., of G .

\Leftarrow Let D' be a refutation of G in \mathbf{K}_{tab} . By the Bisimulation Lemma there exists a refutation D in \mathbb{P}_{tab}^G whose formula image is D' . Evidently, D is a refutation of ϵ . □

7.2. Proof-theoretic properties of the tableau path calculus

We will now prove several statements about properties of derivations in the tableau path calculus \mathbb{P}_{tab}^G , and then use them to find redundancies in the inverse path calculus defined below in Section 7.3.

Let us call the **modal length** of a path the number of occurrences of \Diamond and \Box in the path. For example, the modal length of $\Box \wedge_l \Diamond \Diamond \vee_r$ is 3.

7.5. LEMMA. *Let Π be a path sequent occurring in a derivation of ϵ in \mathbb{P}_{tab}^G . Then all paths in Π have the same modal length.*

PROOF. First, observe the following fact. Let an inference in \mathbb{P}_{tab}^G have a conclusion Π satisfying the claim. Then all premises of this inferences satisfy the claim. Second, apply a straightforward induction on the number of inferences. □

A path π' is called a **prefix** of a path π , denoted $\pi' \sqsubseteq \pi$, if $\pi = \pi' \pi''$ for some path π'' . Evidently, if $\pi' \sqsubseteq \pi$ and π is a G -path, then π' is a G -path and $G|_{\pi'}$ is a subformula of $G|_{\pi}$. A prefix π' of π is called **proper**, denoted $\pi' \sqsubset \pi$, if $\pi' \neq \pi$. A path sequent Π is called **prefix-free** if for every pair π_1, π_2 of different occurrences of paths in Π , π_1 is not a prefix of π_2 . Note that a prefix-free sequent cannot contain two occurrences of the same path.

7.6. LEMMA (Prefix-Freedom). *Any path sequent occurring in a derivation of ϵ is prefix-free.*

PROOF. As in the proof of Lemma 7.5, observe that this property holds for all premises of an inference, whenever it holds for its conclusion. □

We say that two paths π_1 and π_2 form a **\vee -fork** if one of the following conditions is satisfied:

1. $\pi_1 = \pi \vee_l \pi'_1$, $\pi_2 = \pi \vee_r \pi'_2$ for some paths π, π'_1, π'_2 ; or, symmetrically,
2. $\pi_1 = \pi \vee_r \pi'_1$, $\pi_2 = \pi \vee_l \pi'_2$ for some paths π, π'_1, π'_2 .

The paths π_1 and π_2 are called **diamond-separated** if $\pi_1 = \pi'_1 \Diamond \pi''_1$ and $\pi_2 = \pi'_2 \Diamond \pi''_2$ for some paths $\pi'_1, \pi'_2, \pi''_1, \pi''_2$ such that the paths π'_1 and π'_2 have the same modal length and such that $\pi''_1 \neq \pi''_2$.

For example, consider the paths of Figure 23. The paths $\wedge_r \wedge_l \square \vee_l$ and $\wedge_r \wedge_l \square \vee_r$ form a \vee -fork. The paths $\wedge_r \wedge_r \wedge_l \diamond$ and $\wedge_r \wedge_r \wedge_r \diamond$ are diamond-separated.

7.7. LEMMA. *Let Π be a path sequent occurring in a derivation of ϵ in \mathbb{P}_{tab}^G . Then no pair of paths π_1, π_2 in Π forms a \vee -fork.*

PROOF. The proof is by contradiction. We assume that two paths π_1, π_2 occurring in Π form an \vee -fork. Without loss of generality we assume that $\pi_1 = \pi \vee_l \pi'_1$ and $\pi_2 = \pi \vee_r \pi'_2$ for some paths π, π'_1, π'_2 .

Consider the branch \mathcal{B} of the derivation on which Π lies and the topmost sequents on \mathcal{B} where $\pi \vee_l$ and $\pi \vee_r$ first appeared. Without loss of generality assume that $\pi \vee_l$ first appeared above $\pi \vee_r$ (the other case is symmetric). Then the derivation contains the subderivation

$$\frac{\frac{\Gamma_1, \pi \vee_l \quad \Gamma_1, \pi \vee_r}{\Gamma_1, \pi} (\vee)}{\frac{\Gamma_2, \pi \vee_l \quad \Gamma_2, \pi \vee_r}{\Gamma_2, \pi} (\vee)}.$$

Since Γ_1, π is above $\Gamma_2, \pi \vee_r$ in the derivation, Γ_2 must contain a prefix of π . This contradicts Lemma 7.6, since the path sequent $\Gamma_2, \pi \vee_r$ should be prefix-free. \square

7.8. LEMMA. *Let Π be a path sequent occurring in a derivation of ϵ in \mathbb{P}_{tab}^G . Then no pair of paths π_1, π_2 in Π is diamond-separated.*

PROOF. The proof is by contradiction. We assume that a pair of paths π_1, π_2 occurring in Π is diamond-separated. Then $\pi_1 = \pi'_1 \diamond \pi''_1$, $\pi_2 = \pi'_2 \diamond \pi''_2$, $\pi'_1 \neq \pi'_2$, and paths π'_1 and π'_2 have the same modal length L .

Consider the branch \mathcal{B} of derivation on which Π lies and the topmost sequents on \mathcal{B} where $\pi'_1 \diamond$ and $\pi'_2 \diamond$ first appeared. Without loss of generality assume that $\pi'_1 \diamond$ first appeared above $\pi'_2 \diamond$ (the other case is symmetric). Then the derivation contains a subderivation

$$\frac{\frac{\Gamma_1 \square, \pi'_1 \diamond}{\Gamma_1, \pi'_1, \Delta_1} (\diamond)}{\frac{\Gamma_2 \square, \pi'_2 \diamond}{\Gamma_2, \pi'_2, \Delta_2} (\diamond)}.$$

Since π'_2 has the modal length L , it is not hard to argue that all path sequents between $\Gamma_2 \square, \pi'_2 \diamond$ and $\Gamma_1, \pi'_1, \Delta_1$ contain a formula with the modal length at least $L+1$. Therefore, the sequent $\Gamma_1, \pi'_1, \Delta_1$ contains a path of the modal length at least $L+1$ and the path π'_1 with the modal length L . This contradicts Lemma 7.5, since all paths in a path sequent must have the same modal length. \square

In view of Lemmas 7.7 and 7.8 we say that a pair of paths π_1, π_2 are *in conflict* if they either form a \vee -fork or are diamond-separated.

The lemmas above assert some properties of all possible derivations in the path calculus. Now we want to restrict search for a refutation to only a subset of derivations by introducing an ordering on the set of all G -paths.

In ordered resolution for classical logic, one can order literals in clauses and require that a resolution inference be only applied when the greatest literals in both clauses are resolved. We want to introduce a similar restriction in the modal case, by modifying classical literal orderings to their modal equivalent, called a G -ordering. In the classical case, we can use *any* ordering on subformulas of G (or on G -paths) that respects the prefix relation. In the modal case, the situation is more complex, as we will now see: not every ordering on paths preserves completeness. For example, consider the top inference of the derivation of Figure 25:

$$\frac{\begin{array}{c} \Lambda_l \Box, \Lambda_r \Lambda_l \Box \vee_l, \\ \Lambda_r \Lambda_r \Lambda_l \Diamond \end{array} \quad \begin{array}{c} \Lambda_l \Box, \Lambda_r \Lambda_l \Box \vee_r, \\ \Lambda_r \Lambda_r \Lambda_l \Diamond \end{array}}{\Lambda_l \Box, \Lambda_r \Lambda_l \Box, \Lambda_r \Lambda_r \Lambda_l \Diamond} (\vee).$$

It is essential in this derivation that an (\vee) -inference is applied above the rule (\Diamond) , because the rule (\Diamond) is not applicable to the top sequents. However, if we impose an ordering on paths in which $\Lambda_r \Lambda_l \Box \vee_l$ is the smallest path in the first premise, we will rule out the possibility of applying (\vee) first, and no proof will be found. So, the definition of G -ordering is more complex than in the classical case.

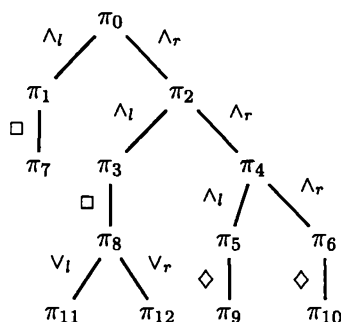
We call two paths *brothers* if one of them has the form $\pi \Lambda_l$ and the other $\pi \Lambda_r$, or likewise, if one of them has the form $\pi \vee_l$ and the other $\pi \vee_r$. For example, the following two paths in our example formula E are brothers: $\Lambda_r \Lambda_l \Box \vee_l$ and $\Lambda_r \Lambda_l \Box \vee_r$, the formulas on this path are the two immediate subformulas of the disjunction $\neg a \vee b$. In general, every conjunction and disjunction in a formula gives rise to a pair of brothers.

We denote by the generic symbol \bowtie any of the symbols \wedge, \vee . Likewise, we denote by $*$ any of the symbols l, r and use notation like Λ_* or \vee_* . Thus \bowtie_* denotes any of the symbols $\Lambda_l, \Lambda_r, \vee_l, \vee_r$. We also denote by the generic symbol \Diamond any of the symbols \Diamond, \Box .

Let us fix a formula G . We call a G -ordering any total order \succ on the set of all G -paths satisfying the following conditions:

1. $\pi_1 \succ \pi_2$, whenever
 - (a) the modal length of π_1 is strictly greater than the modal length of π_2 ; or
 - (b) π_1, π_2 have the same modal length, the last symbol of π_1 is \bowtie_* , and the last symbol of π_2 is \Diamond ; or
 - (c) π_1, π_2 have the same modal length and $\pi_2 \sqsubset \pi_1$. (Note that this case and the previous one are not exclusive, but they do not contradict each other.)
2. There is no path between two brothers, i.e., there exists no G -paths π_1, π_2, π_3 such that $\pi_1 \succ \pi_2 \succ \pi_3$ and π_1, π_3 are brothers.

The intuitive meaning of \succ is that we should first apply rules to the formulas greater with respect to \succ . The relation \succ is also defined in such a way that the

Figure 26: A E -ordering

conclusion of any rule is less than both its premises in the multiset ordering on sequents induced by \succ . Condition 1a ensures that the conclusion is less than the premise when the rules (\Diamond) or (\Diamond^+) are applied. Condition 1b is due to the fact that an application of (\Diamond) or (\Diamond^+) to a sequent containing a path $\pi\mathbb{X}_*$ may give an incomplete calculus. Conditions 1c and 2 are more technical and used in some proofs of this section.

It will be clear why these conditions are required of G -orderings when we prove Lemma 7.12 below.

To illustrate G -orderings, we will show one possible E -ordering (recall that E is our example goal formula). The paths in E were illustrated in Figure 23. In Figure 26, we show these paths again; this time each path is labeled as π_i , where i is a natural number. We claim that the ordering given by $\pi_i \succ \pi_j$ if $i > j$ is an E -ordering. Indeed, the conditions (1a)–(1c) of G -orderings are easy to check. We leave it to the reader to verify that condition (2) holds, too.

We write $\pi \succeq \pi'$ to express that $\pi \succ \pi'$ or $\pi = \pi'$.

In addition to having more complex conditions on orderings for the modal case, another problem will appear in the proof of completeness. For classical logic, one can prove completeness of ordered resolution by purely semantical considerations. In the modal case, we want also to prove that the strategy of selecting the greatest formula (or path) in a clause is compatible with the redundancies considered so far, so we will follow the same pattern of proof: first prove a property of derivations in the tableau calculus, and then transfer it to the inverse calculus using a variant of the Subsequent Lemma. However, unlike the redundancies considered so far, such a proof of completeness will meet a technical obstacle whose essence can be expressed shortly as follows: *the ordering requirement is formulated in terms of premises of an inference, while the completeness proofs for the tableau calculus expand proofs from conclusions*. This obstacle leads to a slightly complex definition of the ordering on paths.

We will now prove that G -orderings exist for every G . To this end we will describe an **ordering procedure** that orders G -paths and note that the resulting ordering

```

Initially, define  $S_i$  to be the set of paths in  $G$  of modal length  $i$ ; /* 1 */
for all  $S_i$  except the last one do /* 2 */
  let  $\pi_1, \dots, \pi_n$  be all paths in  $S_i$  ending at  $\emptyset$ ; /* 3 */
  split  $S_i$  into  $S_i - \{\pi_1, \dots, \pi_n\} \succ \{\pi_1\} \succ \dots \succ \{\pi_n\}$ ; /* 4 */
split  $S_0$  into  $S_0 - \{\epsilon\} \succ \{\epsilon\}$ ; /* 5 */
while there exists  $S_i$  with more than one member do /* 6 */
  let  $\pi\mathbb{X}_l, \pi\mathbb{X}_r$  be two brothers in  $S_i$  such that  $\pi \notin S_i$ ; /* 7 */
  let  $L = \{\pi' \mid \pi' \in S_i \text{ and } \pi\mathbb{X}_l \sqsubset \pi'\}$ ;
  let  $R = \{\pi' \mid \pi' \in S_i \text{ and } \pi\mathbb{X}_r \sqsubset \pi'\}$ ; /* 8 */
  split  $S_i$  into  $S_i - (R \cup L \cup \{\pi\mathbb{X}_r, \pi\mathbb{X}_l\}) \succ R \succ L \succ \{\pi\mathbb{X}_r\} \succ \{\pi\mathbb{X}_l\}$ ; /* 9 */

```

Figure 27: An ordering procedure

is a G -ordering.

The procedure works with a sequence of sets of paths; this sequence is denoted $S_n \succ S_{n-1} \succ \dots \succ S_0$. The intuitive meaning of the sequence is that for all $\pi \in S_i$ and $\pi' \in S_{i-1}$ we have $\pi \succ \pi'$. The paths belonging to the same S_i are yet unordered (but will be ordered later). At each step we will pick up one set S_i in the sequence containing one or more members and replace S_i by two or more sets $S_{i1} \succ \dots \succ S_{ik}$ such that $S_{i1} \cup \dots \cup S_{ik} = S_i$. The procedure terminates when each set contains exactly one member. An ordering procedure is shown in Figure 27. Note that some sets obtained by the procedure (for example L or R) may be empty; in this case we do not include them in the sequence. When the procedure terminates, the sequence consists of singleton sets. We then let $\pi \succ \pi'$ if the sequence has the form $\dots \{\pi\} \succ \dots \{\pi'\} \dots$.

We show how the E -ordering of Figure 26 can be obtained by this procedure. The splitting steps will be denoted by \rightarrow superscribed by the line number in the algorithm. Instead of singleton sets $\{\pi_i\}$ we write π_i .

```

 $\{\pi_7, \pi_8, \pi_9, \pi_{10}, \pi_{11}, \pi_{12}\} \succ \{\pi_0, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6\} \rightarrow^4$ 
 $\{\pi_{11}, \pi_{12}\} \succ \pi_{10} \succ \pi_9 \succ \pi_8 \succ \pi_7 \succ \{\pi_0, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6\} \rightarrow^5$ 
 $\{\pi_{11}, \pi_{12}\} \succ \pi_{10} \succ \pi_9 \succ \pi_8 \succ \pi_7 \succ \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6\} \succ \pi_0 \rightarrow^9$ 
 $\pi_{12} \succ \pi_{11} \succ \pi_{10} \succ \pi_9 \succ \pi_8 \succ \pi_7 \succ \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6\} \succ \pi_0 \rightarrow^9$ 
 $\pi_{12} \succ \pi_{11} \succ \pi_{10} \succ \pi_9 \succ \pi_8 \succ \pi_7 \succ \{\pi_3, \pi_4, \pi_5, \pi_6\} \succ \pi_2 \succ \pi_1 \succ \pi_0 \rightarrow^9$ 
 $\pi_{12} \succ \pi_{11} \succ \pi_{10} \succ \pi_9 \succ \pi_8 \succ \pi_7 \succ \{\pi_5, \pi_6\} \succ \pi_4 \succ \pi_3 \succ \pi_2 \succ \pi_1 \succ \pi_0 \rightarrow^9$ 
 $\pi_{12} \succ \pi_{11} \succ \pi_{10} \succ \pi_9 \succ \pi_8 \succ \pi_7 \succ \pi_6 \succ \pi_5 \succ \pi_4 \succ \pi_3 \succ \pi_2 \succ \pi_1 \succ \pi_0.$ 

```

7.9. LEMMA. *Every ordering produced by an ordering procedure on G is a G -ordering.*

The proof can be found in [Voronkov 2001].

Since any run of the ordering procedure results in a G -ordering, we obtain the following corollary.

7.10. COROLLARY. G -orderings exist. □

The notion of G -ordering is introduced in order to prove the existence of refutations of a special form related to these orderings and therefore to restrict the search space to such refutations only.

For a path π and a path sequent Γ we write $\pi \succ \Gamma$ to denote $\pi \succ \pi'$ for every π' in Γ .

A (\vee) -inference in \mathbb{P}_{tab}^G

$$\frac{\Gamma, \pi \vee_l \quad \Gamma, \pi \vee_r}{\Gamma, \pi} (\vee)$$

is said to **respect a G -ordering** \succ if $\pi \vee_l \succ \Gamma$ and $\pi \vee_r \succ \Gamma$. Likewise, a (\wedge) -inference in \mathbb{P}_{tab}^G

$$\frac{\Gamma, \pi \wedge_l, \pi \wedge_r}{\Gamma, \pi} (\wedge)$$

is said to respect \succ if $\pi \wedge_l \succ \Gamma$ and $\pi \wedge_r \succ \Gamma$. A derivation in \mathbb{P}_{tab}^G is said to respect \succ if every (\vee) and every (\wedge) -inference of this derivation respect \succ .

7.11. LEMMA. *Let G be a unsatisfiable formula and \succ be a G -ordering. Then there exists a refutation of ϵ in \mathbb{P}_{tab}^G that respects \succ .*

It is not immediately obvious how to prove this lemma. If we try to prove this lemma by induction, starting with the empty derivation of the empty path and trying to extend the derivation from conclusion to premises, we may be in a “deadlock” situation: i.e., obtain a path sequent Γ such that *every* (\wedge) - and (\vee) -inference having Γ as the conclusion does not respect \succ . Let us illustrate such a deadlock situation. Suppose that $\pi_1 \Box$ and $\pi_2 \vee_l$ are G -paths such that π_1, π_2 have the same modal length and $\pi_1 \succ \pi_2 \vee_l$. Consider the sequent π_1, π_2 . We can apply a (\vee) -inference in the tableau path calculus to obtain this sequent:

$$\frac{\pi_1, \pi_2 \vee_l \quad \pi_1, \pi_2 \vee_r}{\pi_1, \pi_2} (\vee).$$

In fact, this inference is *the only* way to derive π_1, π_2 . However, this inference does not respect \succ .

The conditions on an inference to respect \succ are formulated in terms of the *premises* of the inference, but in such an inductive proof we can only extend the *conclusion* by a new inference. To overcome this problem, we will try to avoid path sequents that bring us into a deadlock, by restricting ourselves to the so-called \succ -compact path sequents defined below.

Let π_1, \dots, π_n be paths in G and \succ be a G -ordering. The path sequent $\Gamma = \pi_1, \dots, \pi_n$ is called \succ -compact if for every $i = 1, \dots, n$,

1. if π_i is a \wedge -path, then $\pi_i \wedge_l \succ \Gamma$ and $\pi_i \wedge_r \succ \Gamma$;
2. likewise, if π_i is a \vee -path, then $\pi_i \vee_l \succ \Gamma$ and $\pi_i \vee_r \succ \Gamma$.

Using our generic notation, we can reformulate the definition of \succ -compactness as “for every \mathbb{X} -path π_i in Γ we have $\pi_i \mathbb{X}_* \succ \Gamma$.” Note that $\pi_i \mathbb{X}_* \succ \pi_i$ is guaranteed by the conditions on G -orderings, therefore the requirement $\pi_i \mathbb{X}_* \succ \Gamma$ can be replaced by $\pi_i \mathbb{X}_* \succ \Gamma \setminus \{\pi_i\}$.

The next lemma asserts that a compact sequent Γ cannot lead to a deadlock.

7.12. LEMMA. *Let \succ be a G -ordering. Then*

1. *the premise of every (\Diamond) -inference is \succ -compact.*
Furthermore, let Γ be a \succ -compact sequent occurring in a derivation of ϵ . Then
2. *every (\mathbb{X}) -inference having Γ as the conclusion, respects \succ ;*
3. *if Γ contains at least one \wedge -path or \vee -path, then there exists a (\mathbb{X}) -inference all whose premises are \succ -compact.*

PROOF.

1. We prove: *the premise of every (\Diamond) -inference*

$$\frac{\Pi\Box, \pi\Diamond}{\Gamma, \Pi, \pi} (\Diamond)$$

is \succ -compact, i.e. for every π' in the premise, if $\pi' \mathbb{X}_*$ is a G -path, then $\pi' \succ \Pi\Box, \pi\Diamond - \{\pi'\}$. Let us prove $\pi\Diamond \mathbb{X}_* \succ \Pi\Box$, the case when $\pi' \in \Pi\Box$ is similar. By Lemma 7.5, all paths in $\Pi\Box, \pi\Diamond$ are of the same modal length, so $\pi\Diamond \mathbb{X}_*$ is of the same modal length as any path in $\Pi\Box$. By condition 1b of the definition of G -ordering we have $\pi\Diamond \mathbb{X}_* \succ \Pi\Box$.

2. We prove: *every (\mathbb{X}) -inference having Γ as the conclusion, respects \succ* . Consider the case of (\wedge) -inferences, the case of (\vee) is similar. Then Γ is of the form Π, π and the inference is

$$\frac{\Pi, \pi\wedge_l, \pi\wedge_r}{\Pi, \pi} (\wedge).$$

Since Π, π is \succ -compact, $\pi\wedge_l \succ \Pi, \pi$ and $\pi\wedge_r \succ \Pi, \pi$, therefore the inference respects \succ .

3. We assume that Γ contains at least one \mathbb{X} -path and prove: *there exists a (\mathbb{X}) -inference all whose premises are \succ -compact*. Consider the set of paths

$$S = \{\pi \mathbb{X}_* \mid \pi \in \Gamma \text{ and } \pi \mathbb{X}_* \text{ is a } G\text{-path}\}.$$

By the assumption, this set is nonempty. Since \succ is a linear ordering, this set has the least element with respect to \succ . Without loss of generality we assume that this path is $\pi\wedge_l$. Then Γ has the form Π, π for some path sequent Π . Consider the inference

$$\frac{\Pi, \pi\wedge_l, \pi\wedge_r}{\Pi, \pi} (\wedge).$$

We claim that the premise of this inference is \succ -compact. This amounts to proving that (3a) $\pi \wedge_* \mathbb{X}_* \succ \Pi$, whenever $\pi \wedge_* \mathbb{X}_*$ is a G -path; (3b) for every path $\pi' \in \Pi$ we have $\pi' \mathbb{X}_* \succ \Pi, \pi \wedge_*$, whenever $\pi' \mathbb{X}_*$ is a G -path; and (3c) $\pi \wedge_l \mathbb{X}_* \succ \pi \wedge_r$ and $\pi \wedge_l \mathbb{X}_* \succ \pi \wedge_r$. Let us prove this.

- (a) Since Π, π is \succ -compact, $\pi \wedge_* \succ \Pi$. Evidently, $\pi \wedge_*$ is a prefix of $\pi \wedge_* \mathbb{X}_*$; thus by the definition of G -orderings (condition 1c) we have $\pi \wedge_* \mathbb{X}_* \succ \pi \wedge_*$. This implies $\pi \wedge_* \mathbb{X}_* \succ \Pi$.
- (b) Since Π, π is compact, $\pi' \mathbb{X}_* \succ \Pi$, so it remains to prove $\pi' \mathbb{X}_* \succ \pi \wedge_*$, i.e. $\pi' \mathbb{X}_* \succ \pi \wedge_l$ and $\pi' \mathbb{X}_* \succ \pi \wedge_r$. Note that $\pi' \mathbb{X}_* \in S$ and that $\pi \wedge_l$ is the least element of S , therefore $\pi' \mathbb{X}_* \succeq \pi \wedge_l$. By Prefix-Free Lemma 7.6, $\pi' \mathbb{X}_*$ and $\pi \wedge_l$ cannot coincide, thus $\pi' \mathbb{X}_* \succ \pi \wedge_l$. Again by this lemma, $\pi' \mathbb{X}_*$ cannot be a prefix of $\pi \wedge_l$. In addition, we have $\pi' \mathbb{X}_* \succ \pi \wedge_l$, and $\pi \wedge_l$ is a brother to $\pi \wedge_r$, then by condition 2 of the definition of G -ordering, $\pi' \mathbb{X}_* \succ \pi \wedge_r$.
- (c) We will only prove $\pi \wedge_l \mathbb{X}_* \succ \pi \wedge_r$, the second statement is symmetric. Suppose, by contradiction, $\pi \wedge_r \succ \pi \wedge_l \mathbb{X}_*$. Since \succ respects the prefix relation, we also have $\pi \wedge_r \succ \pi \wedge_l \mathbb{X}_* \succ \pi \wedge_l$. This contradicts condition 2 of the definition of G -orderings, since the path $\pi \wedge_l \mathbb{X}_*$ is between two brothers.

□

This proof explains the conditions used in the definition of G -orderings. The only condition that has not been used is 1a, and indeed this condition can be omitted. This condition ensures that in the multiset ordering on sequents induced by \succ the premise of any rule of \mathbf{K}_{tab} is greater than its conclusion.

Now we can prove Lemma 7.11: if G is unsatisfiable, then there exists a refutation of ϵ in \mathbb{P}_{tab}^G that respects \succ .

PROOF (of Lemma 7.11). The lemma is proved by induction. Starting with the path sequent ϵ , we will construct, step by step, a derivation D of ϵ such that (i) D consists only of \succ -compact path sequents, (ii) the formula images of all top sequents of D are unsatisfiable; until we obtain a refutation. Lemma 7.12 guarantees that the obtained refutation respects \succ . At every step we choose any top path sequent Π of D that is not an axiom and extend the derivation by adding an inference having Π as its conclusion.

1. (Induction base.) *The sequent ϵ is \succ -compact.* This is obvious.
2. (Induction step). Denote by Γ the formula image of Π . We have that Γ is unsatisfiable and not an axiom, and Π is \succ -compact. We show how to find an inference with the conclusion Π and premises satisfying the same properties. If Γ contains any conjunction or disjunction, then by Lemma 7.12 there exists an (\mathbb{X}) -inference having Π as the conclusion and \succ -compact premises. Since (\mathbb{X}) -rules are invertible (Lemma 7.1), the formula images of these premises are also unsatisfiable.

If Γ contains neither conjunction nor disjunction and Γ is unsatisfiable, then there is a (\Diamond) -inference in \mathbf{K}_{tab} with the conclusion Γ and unsatisfiable premise Γ' . It is not hard to argue that there exists an (\Diamond) -inference in \mathbb{P}_{tab}^G with a

$$\begin{array}{c}
\text{axioms: } \pi_1, \pi_2 \quad \frac{\Gamma, \pi, \pi}{\Gamma, \pi} (C) \\
\frac{\Gamma, \pi \wedge_l}{\Gamma, \pi} (\wedge_l) \quad \frac{\Gamma, \pi \wedge_r}{\Gamma, \pi} (\wedge_r) \\
\frac{\Gamma, \pi \vee_l \quad \Delta, \pi \vee_r}{\Gamma, \Delta, \pi} (\vee) \\
\frac{\Gamma \Box, \pi \Diamond}{\Gamma, \pi} (\Diamond) \quad \frac{\Gamma \Box}{\Gamma, \pi} (\Diamond^+)
\end{array}$$

All paths occurring in path sequents are G -paths. In the rule (ax) we have $G|_{\pi_1} = p$ and $G|_{\pi_2} = \neg p$ for some propositional variable p . In the rule (\Diamond^+) , π is an \Diamond -path.

Figure 28: Inverse path calculus \mathbb{P}_{inv}^G

$$\begin{array}{c}
\frac{\wedge_l \Box, \wedge_r \wedge_l \Box \vee_l \quad \wedge_r \wedge_l \Box \vee_r, \wedge_r \wedge_r \wedge_l \Diamond}{\wedge_l \Box, \wedge_r \wedge_l \Box, \wedge_r \wedge_r \wedge_l \Diamond} (\vee) \\
\frac{\wedge_l \Box, \wedge_r \wedge_l \Box, \wedge_r \wedge_r \wedge_l \Diamond}{\wedge_l, \wedge_r \wedge_l, \wedge_r \wedge_r \wedge_l} (\Diamond) \\
\frac{\wedge_l, \wedge_r \wedge_l, \wedge_r \wedge_r \wedge_l}{\wedge_l, \wedge_r, \wedge_r} (\wedge), (\wedge), (\wedge) \\
\frac{\wedge_l, \wedge_r, \wedge_r}{\wedge_l, \wedge_r} (C) \\
\frac{\wedge_l, \wedge_r}{\epsilon} (\wedge), (\wedge), (C)
\end{array}$$

Some sequences of inferences are shown as one inference.

Figure 29: Derivation in \mathbb{P}_{inv}^G

premise Π' such that Γ' is the formula image of Π' . By Lemma 7.12, Π' is \succ -compact.

The termination of this construction follows from the following observation: every infinite construction would yield an infinite derivation in \mathbf{K}_{tab} which does not exist.

□

7.3. Inverse path calculus

In this section we define the inverse version of the path calculus, obtained by turning \mathbf{K}_{inv}^G into a path calculus.

The **inverse path calculus** \mathbb{P}_{inv}^G and a refutation in it are given in Figures 28 and 29.

The same arguments as in Bisimulation Lemma 7.3 prove the following lemma.

7.13. LEMMA (Bisimulation Lemma for \mathbb{P}_{inv}^G). (i) Let D be a derivation in \mathbb{P}_{inv}^G . Then the formula image of D is a derivation of G in \mathbf{K}_{inv}^G . (ii) Let D' be a derivation

of a sequent A_1, \dots, A_n in K_{inv}^G , and π_1, \dots, π_n be paths such that $G|_{\pi_i} = A_i$, for all $i = 1, \dots, n$. Then there exists a derivation D of π_1, \dots, π_n in \mathbb{P}_{inv}^G such that D' is the formula image of D . (iii) Both (i) and (ii) also hold if “derivation” is replaced by “refutation” everywhere. \square

The calculus \mathbb{P}_{inv}^G has different rules for handling \wedge , so we change the definition of derivations respecting a G -ordering in the following way.

A derivation in \mathbb{P}_{inv}^G is said to **respect a G -ordering** \succ if

1. for every (\vee) inference of this derivation

$$\frac{\Gamma, \pi \vee_l \quad \Delta, \pi \vee_r}{\Gamma, \Delta, \pi} (\vee)$$

we have $\pi \vee_l \succ \Gamma$ and $\pi \vee_r \succ \Delta$;

2. likewise, for every (\wedge_l) inference (respectively (\wedge_r) inference) of this derivation

$$\frac{\Gamma, \pi \wedge_l}{\Gamma, \pi} (\wedge_l) \quad (\quad \text{respectively} \quad \frac{\Gamma, \pi \wedge_r}{\Gamma, \pi} (\wedge_r) \quad)$$

we have $\pi \wedge_l \succ \Gamma$ (respectively $\pi \wedge_r \succ \Gamma$).

The next theorem is the main theorem of this section. It asserts that a formula is unsatisfiable if and only if it has a refutation satisfying several redundancy criteria. Namely, we prove that any sequent containing a conflicting pair is redundant, and that any inference not respecting \succ is redundant as well. Later, we will give an even stronger version of this statement related to the subsumption rule, but we will need to introduce a new notion of derivation first.

We denote by $\mathbb{P}_{inv}^{G, \succ}$ the calculus obtained from \mathbb{P}_{inv}^G by (i) removal of all path sequents containing paths of different modal lengths, (ii) removal of all path sequents containing a conflicting pair of paths, (iii) removal of all inferences that do not respect \succ .

7.14. THEOREM (Completeness of $\mathbb{P}_{inv}^{G, \succ}$). *A formula G is unsatisfiable if and only if ϵ has a refutation in the calculus $\mathbb{P}_{inv}^{G, \succ}$.*

The key lemma in the proof of Theorem 7.14 is as follows.

7.15. LEMMA (Subsequent Lemma for $\mathbb{P}_{inv}^{G, \succ}$). *Suppose that D is a refutation of ϵ in \mathbb{P}_{tab}^G that respects \succ and I is an inference in D of the form*

$$\frac{\Gamma_1 \quad \dots \quad \Gamma_n}{\Gamma},$$

and path sequents $\Delta_1, \dots, \Delta_n$ are such that each Δ_i is a subsequent of Γ_i . Then there exists a derivation D' of Δ in $\mathbb{P}_{inv}^{G, \succ}$ from $\Delta_1, \dots, \Delta_n$ such that Δ is a subsequent of Γ .

PROOF. We will give a proof by case analysis on the inference I , after having noted one general property of $\Delta, \Delta_1, \dots, \Delta_n$.

By Lemmas 7.7 and 7.8 each path sequent among $\Gamma_1, \dots, \Gamma_n$ contains no conflicting pair of paths. Since $\Delta_1, \dots, \Delta_n$ are submultisets of $\Gamma_1, \dots, \Gamma_n$, respectively, we obtain that none of $\Delta_1, \dots, \Delta_n$ contains a conflicting pair of paths. Likewise, if we find a derivation of Δ from $\Delta_1, \dots, \Delta_n$, since Δ is a submultiset of Γ , we obtain that Δ contains no conflicting pair of paths. In a similar way we can check that it contains no path sequent with a pair of paths of different modal length. Therefore, it is enough to check that the intermediate sequents of D' do not violate the condition, which will be obvious in all cases.

Therefore, we will only show how to find the derivation D' and prove that it respects \succ . Consider all possible cases of inferences in \mathbb{P}_{tab}^G .

1. The inference I is an axiom Γ, π_1, π_2 such that $G|_{\pi_1} = \neg G|_{\pi_2}$. Then π_1, π_2 is a subsequence of Γ, π_1, π_2 , and is an axiom of $\mathbb{P}_{inv}^{G, \succ}$. Therefore, we can let D' be the derivation consisting of one axiom π_1, π_2 . Evidently, it respects \succ .
2. The inference I has the form

$$\frac{\Pi, \pi \wedge_l, \pi \wedge_r}{\Pi, \pi} (\wedge).$$

Note that $\pi \wedge_l, \pi \wedge_r \succ \Pi$, because D respects \succ . We assume that some sequent Δ_1 is a subsequence of $\Pi, \pi \wedge_l, \pi \wedge_r$ and find a derivation in \mathbb{P}_{inv}^G of a subsequence of Π, π from Δ_1 .

Consider the following possible cases:

- (a) *Case: Δ_1 contains neither $\pi \wedge_l$, nor $\pi \wedge_r$.* In this case Δ_1 is a subsequence of Π . Then Δ_1 is also a subsequence of Π, π , and the empty derivation of Δ_1 from Δ_1 proves the case.
- (b) *Case: Δ_1 contains $\pi \wedge_l$, but not $\pi \wedge_r$.* In this case $\Delta_1 = \Delta', \pi \wedge_l$ for some subsequence Δ' of Π . The following derivation proves the case:

$$\frac{\Delta', \pi \wedge_l}{\Delta', \pi} (\wedge_l).$$

Since $\pi \wedge_l, \pi \wedge_r \succ \Pi$ and Δ' is a submultiset of Π , we have $\pi \wedge_l \succ \Delta'$. The ordering condition can be checked in the similar way for all other cases, we do not consider it any more.

- (c) *Case: Δ_1 contains $\pi \wedge_r$, but not $\pi \wedge_l$.* This case is symmetric to the previous one.
- (d) *Case: Δ_1 contains both $\pi \wedge_l$ and $\pi \wedge_r$.* In this case $\Delta_1 = \Delta', \pi \wedge_l, \pi \wedge_r$ for some submultiset Δ' of Π . Without loss of generality assume $\pi \wedge_l \succ \pi \wedge_r$ (and hence $\pi \wedge_l \succ \Delta', \pi \wedge_r$). The following derivation proves the case:

$$\begin{array}{c} \frac{\Delta', \pi \wedge_l, \pi \wedge_r}{\Delta', \pi, \pi \wedge_r} (\wedge_l) \\ \frac{\Delta', \pi, \pi \wedge_r}{\Delta', \pi, \pi} (\wedge_r) \\ \frac{\Delta', \pi, \pi}{\Delta', \pi} (C). \end{array}$$

Other cases are considered in [Voronkov 2001]. □

By induction on the length of derivations one can easily generalize Lemma 7.15 from inferences to arbitrary derivations:

7.16. LEMMA. *Let D be a refutation of ϵ in \mathbb{P}_{tab}^G that respects \succ , D'' be a subderivation in D of a sequent Γ from sequents $\Gamma_1, \dots, \Gamma_n$, and $\Delta_1, \dots, \Delta_n$ be subsequents of $\Gamma_1, \dots, \Gamma_n$, respectively. Then there exists a derivation D' of Δ in $\mathbb{P}_{inv}^{G, \succ}$ from $\Delta_1, \dots, \Delta_n$ such that Δ is a subsequence of Γ .* □

Now we can prove Theorem 7.14.

PROOF (of Theorem 7.14).

- (\Rightarrow) Suppose that a formula G is unsatisfiable. By Completeness Theorem 7.4 for \mathbb{P}_{tab}^G , the sequent ϵ has a refutation in \mathbb{P}_{tab}^G . By Lemma 7.16, there exists a subsequence of ϵ that has a refutation in $\mathbb{P}_{inv}^{G, \succ}$. Evidently, the empty sequent has no refutation in \mathbb{P}_{inv}^G ; therefore ϵ has a refutation in $\mathbb{P}_{inv}^{G, \succ}$.
- (\Leftarrow) Suppose that ϵ has a refutation in $\mathbb{P}_{inv}^{G, \succ}$. By Bisimulation Lemma 7.13 for \mathbb{P}_{inv}^G , the sequent G has a refutation in \mathbb{K}_{inv}^G . It is not hard to argue that \mathbb{K}_{inv}^G is sound, i.e., every sequent that has a refutation is unsatisfiable. Therefore, G is unsatisfiable. □

Let \succ be a G -ordering. Consider the standard **multiset extension** of \succ , denoted \succ^{mul} , defined as follows: $\Pi \succ^{mul} \Pi'$ if $\Pi \neq \Pi'$ and for every $\pi' \in \Pi' - \Pi$ there exists a $\pi \in \Pi - \Pi'$ such that $\pi \succ \pi'$. It is well-known that \succ^{mul} is a well-founded order. Note that all inference rules of $\mathbb{P}_{inv}^{G, \succ}$ have the following property: the conclusion of any rule is strictly less than every premise of this rule with respect to \succ^{mul} . For (\bowtie) rules, this follows from the fact that the conclusion is obtained from the multiset union of the premises by replacing the greatest path by one or more smaller paths. For the contraction rule this is obvious. For (\diamond) and (\diamond^+)-rules, this follows from the first condition of G -orderings. This implies that any derivation process from a finite set of path sequents in $\mathbb{P}_{inv}^{G, \succ}$ will terminate (so we do not even need a subsumption rule that is often essential for termination of bottom-up derivation processes).

7.4. More redundancies: subsumption

We say that a path sequent Π **subsumes** a path sequent Π' if Π is a subsequence of Π' . Saturation-based theorem provers for first-order logic often use subsumption as a redundancy criterion: clauses subsumed by other clauses are removed from the search space.

In order to formalize subsumption as a redundancy criterion, we need a new notion of derivation, since subsumed clauses are not redundant per se, but only

redundant with respect to other clauses in the search space. So to prove completeness of subsumption, we need to formalize the notion of search space. In this section we give such a formalization and prove that subsumption can be used to remove clauses from the search space. The formalization here uses two main ideas: (i) inference rules are reformulated as behaving on sets of clauses, as in [Bachmair and Ganzinger 2001] (Chapter 2 of this Handbook); (ii) a very general property of derivations on such sets is proved to guarantee completeness in the style of Voronkov [1992]. Then we prove that the system with subsumption satisfies these properties.

As before, we assume to deal with *inferences*, consisting of zero or more premises and one conclusion, we assume that all premises and the conclusion belong to a set of derivable objects, or simply *objects*. Examples of objects used in this chapter are sequents and path sequents. An *inference rule* is a collection of inferences. An *inference system* or a *calculus* is a collection of inference rules. Alternatively, we will consider an inference system as consisting of inferences that are instances of inference rules of this system. Suppose that \mathcal{I} is an inference system on objects. We turn it into an inference system $\{\mathcal{I}\}$ on *sets of objects*, called the *set calculus* as follows. Take any inference

$$\frac{\Pi_1 \cdots \Pi_n}{\Pi} \quad (7.2)$$

of \mathcal{I} . Then for every set S of objects the following is an inference rule of $\{\mathcal{I}\}$:

$$\frac{S \cup \{\Pi_1, \dots, \Pi_n\}}{S \cup \{\Pi_1, \dots, \Pi_n, \Pi\}}.$$

The intuitive meaning of $\{\mathcal{I}\}$ is the following: sets of objects formalize search space; the inference rule above means: if Π_1, \dots, Π_n are already proved (belong to the search space), we can add Π to the search space, provided that (7.2) is an inference of \mathcal{I} . Inferences in $\{\mathcal{I}\}$ will be denoted using the \triangleright sign:

$$S \cup \{\Pi_1, \dots, \Pi_n\} \triangleright S \cup \{\Pi_1, \dots, \Pi_n, \Pi\}.$$

Sometimes we will superscribe \triangleright with the name of the inference rule.

We call the *initial set of objects* of $\{\mathcal{I}\}$ the set of *all* axioms of \mathcal{I} . We also assume that associated with \mathcal{I} is an object, called the *goal object*. For example, in all our path calculi the goal object is the sequent ϵ . We call a *derivation* in the set calculus any sequence of inferences:

$$S_0 \triangleright S_1 \triangleright S_2 \triangleright \dots,$$

possibly infinite, where S_0 is the initial set of objects. We say the derivation *succeeds* if some S_i contains a goal object, and *fails* otherwise.

The subsumption strategy that deletes a clause from the search space can now be formalized as the inference rule in the set calculus

$$S \cup \{\Pi_1, \Pi_2\} \triangleright S \cup \{\Pi_1\},$$

provided that Π_1 subsumes Π_2 .

When we have a notion of derivation that deletes objects from the search space, we can have the following problem. Suppose that an inference rule adds an object Π to the search space, and then such a deletion rule removes it. Then we may obtain an infinite failing derivation:

$$S \triangleright S \cup \{\Pi\} \triangleright S \triangleright S \cup \{\Pi\} \triangleright \dots,$$

even when a successful derivation exists. Such a derivation is *unfair*: there have been inferences applicable to S but that were never applied. Let us formalize the notion of fairness.

Consider a derivation $S_0 \triangleright S_1 \triangleright \dots$ in $\{\mathcal{I}\}$. The set $\bigcup_i \bigcap_{j \geq i} S_j$ is called the *limit* of this derivation. It is easy to see that the limit consists of the sequents that remain in the derivation forever after some step. The derivation is called *fair* if, for every inference of \mathcal{I} ,

$$\frac{\Pi_1 \dots \Pi_n}{\Pi},$$

if Π_1, \dots, Π_n belong to the limit of the derivation, then an inference of $\{\mathcal{I}\}$

$$S \cup \{\Pi_1, \dots, \Pi_n\} \triangleright S \cup \{\Pi_1, \dots, \Pi_n, \Pi\}$$

has been applied in the derivation. This means that for some i and S , the set S_i has the form $S \cup \{\Pi_1, \dots, \Pi_n\}$, and S_{i+1} has the form $S \cup \{\Pi_1, \dots, \Pi_n, \Pi\}$.

We will now prove a statement about $\{\mathbb{P}_{inv}^{G, \succ}\}$, i.e., the set inference system obtained from $\mathbb{P}_{inv}^{G, \succ}$. It will be clear from the proof that the statement is very general and applicable to any inference system without the deletion rule.

7.17. THEOREM (Completeness of $\{\mathbb{P}_{inv}^{G, \succ}\}$). *Let G be a formula and \succ be a G -ordering. (i) If some derivation in $\{\mathbb{P}_{inv}^{G, \succ}\}$ succeeds, then G is unsatisfiable. (ii) If G is unsatisfiable, then every fair derivation in $\{\mathbb{P}_{inv}^{G, \succ}\}$ succeeds.*

PROOF. (i) Let a derivation $S_0 \triangleright \dots \triangleright S_n$ succeed. It is easy to see that every $\Pi \in \bigcup_i S_i$ has a refutation in $\mathbb{P}_{inv}^{G, \succ}$. Therefore ϵ has a refutation, and by completeness of $\mathbb{P}_{inv}^{G, \succ}$ (Theorem 7.14), G is unsatisfiable.

(ii) Let $S = S_0 \triangleright S_1 \triangleright \dots$ be a fair derivation. We prove that every path sequent Π that has a refutation in $\mathbb{P}_{inv}^{G, \succ}$ occurs in some S_i . The proof is by induction on the length of the refutation of Π in $\mathbb{P}_{inv}^{G, \succ}$. If the refutation is of length 0, i.e., Π is an axiom, then Π is already a member of S_0 . For refutations of longer lengths, consider the last inference of this refutation:

$$\frac{\Pi_1 \dots \Pi_n}{\Pi}.$$

By the induction hypothesis, every Π_i is a member of some S_{k_i} . Since $\{\mathbb{P}_{inv}^{G, \succ}\}$ does not delete any path sequents, all Π_i belong to the limit of S . Then by fairness of S , Π is a member of some S_m , too. \square

Now we are ready to prove that $\{\mathbb{P}_{inv}^{G,\succ}\}$ with subsumption is complete, i.e., every fair derivation in this system contains ϵ , whenever G is unsatisfiable. In fact, we will define an even stronger notion of subsumption first, and then prove that $\{\mathbb{P}_{inv}^{G,\succ}\}$ is complete with this stronger notion.

We say that a path sequent Π **prefix-subsumes** a path sequent Π' if for every $\pi \in \Pi \dot{-} \Pi'$ there exists a $\pi' \in \Pi' \dot{-} \Pi$ such that $\pi \sqsubset \pi'$. It is easy to see that prefix-subsumption is the standard multiset extension of the ordering \sqsubset , see [Nieuwenhuis and Rubio 2001, page 383] (Chapter 7 of this Handbook) for precise definitions.

The intuition behind the prefix-subsumption is the following. Suppose that a path $\pi_1\pi_2$ occurs in a path sequent Π , occurring in a derivation of ϵ , and π_2 contains no modal operators. If we consider the branch of this derivation between Π and ϵ , there will be a rule that adds π_1 on the branch. Now, if we replace $\pi_1\pi_2$ in Π by π_1 , obtaining a path sequent Π' , one can see that the derivation can be made into a (maybe shorter) derivation of ϵ from Π' . Therefore, replacement of a path by its subpath of the same modal length preserves derivability of ϵ , and sometimes make derivations shorter.

We call $\{\mathbb{P}_{inv}^{G,\succ}\}$ **with prefix-subsumption** the inference system obtained from $\{\mathbb{P}_{inv}^{G,\succ}\}$ by adding the **prefix-subsumption rule**:

$$S \cup \{\Pi, \Pi'\} \triangleright S \cup \{\Pi\},$$

where Π prefix-subsumes Π' and $\Pi \neq \Pi'$.

Note that the notion of fair derivation introduced above only required inferences of the original calculus (*cumulative* inferences) be applied at some step. Now we have new kinds of steps that *delete* sequents from the search space. We will use the old notion of fairness for $\{\mathbb{P}_{inv}^{G,\succ}\}$ with prefix-subsumption, i.e. we do *not* require that prefix-subsumption inferences applicable to sequents in the limit be applied at some step. Thus, any fair derivation in $\{\mathbb{P}_{inv}^{G,\succ}\}$ without prefix-subsumption is also a fair derivation in $\{\mathbb{P}_{inv}^{G,\succ}\}$ with prefix-subsumption.

Let us note some obvious properties of prefix-subsumption:

7.18. LEMMA.

1. Note that Π subsumes Π' if and only if $\Pi \dot{-} \Pi'$ is empty. Therefore, if Π subsumes Π' , then Π also prefix-subsumes Π' .
2. If Π_1 prefix-subsumes Π_2 and Π_2 prefix-subsumes Π_1 , then $\Pi_1 = \Pi_2$.⁷
3. If Π prefix-subsumes Π' and Π' prefix-subsumes Π'' , then Π prefix-subsumes Π'' .
4. If $\pi_1 \sqsubset \pi'$, \dots , $\pi_n \sqsubset \pi'$, and Π prefix-subsumes Π' , then Π, π_1, \dots, π_n prefix-subsumes Π', π' .
5. If $\pi \sqsubseteq \pi'$ and Π prefix-subsumes Π' , then Π, π prefix-subsumes Π', π' .

⁷Using sets instead of multisets in the formulation of the inverse method creates a subtle problem with prefix-subsumption. The path sequents $\pi\pi', \pi$ and $\pi\pi'$ would prefix-subsume each other if we used sets instead of multisets (because $\pi\pi'$ prefix-subsumes $\pi\pi', \pi$, which, in turn, prefix-subsumes $\pi\pi', \pi\pi'$).

6. *There is no infinite sequence Π_0, Π_1, \dots such that Π_{i+1} prefix-subsumes Π_i and $\Pi_{i+1} \neq \Pi_i$.*

Many of these properties immediately follow from the standard facts about the multiset extensions of well-founded orders, see [Nieuwenhuis and Rubio 2001, page 383] (Chapter 7 of this Handbook); for example the prefix-subsumption relation is a well-founded order because so is \sqsubset .

We will note two further properties of \sqsubseteq related to proof-search optimization after proving completeness of $\{\mathbb{P}_{inv}^{G, \succ}\}$ with prefix-subsumption.

7.19. THEOREM (Completeness of $\{\mathbb{P}_{inv}^{G, \succ}\}$ With Prefix-Subsumption). *(i) If any derivation in the system $\{\mathbb{P}_{inv}^{G, \succ}\}$ with prefix-subsumption succeeds, then G is unsatisfiable. (ii) If G is unsatisfiable, then every fair derivation \mathcal{S} in $\{\mathbb{P}_{inv}^{G, \succ}\}$ with prefix-subsumption succeeds.*

PROOF. (i) is proved as in Theorem 7.17 (completeness of $\{\mathbb{P}_{inv}^{G, \succ}\}$). For (ii), we will use Theorem 7.17. Let us show that (ii) immediately follows from

- (7.3) Let \mathcal{T} be a fair derivation in $\{\mathbb{P}_{inv}^{G, \succ}\}$ and a path sequent Π occurs in \mathcal{T} . Then there exists a path sequent Γ that prefix-subsumes Π and belongs to the limit of \mathcal{S} .

Suppose G is unsatisfiable; then by Theorem 7.17 \mathcal{T} contains the path sequent ϵ . Then by (7.3), \mathcal{S} must contain a path sequent that prefix-subsumes ϵ , but there is only one such path sequent: ϵ itself. So, it remains to prove (7.3).

Suppose $\mathcal{S} = S_0 \triangleright S_1 \triangleright \dots$ and denote by S_∞ the limit of \mathcal{S} . We say that a path sequent Π has been *removed* in \mathcal{S} by Π' if some S_{i+1} is $S_i \setminus \{\Pi\}$, and $\Pi' \in S_{i+1}$ prefix-subsumes Π . If Π has been removed by Π_1 , it is possible that Π_1 is later removed by Π_2 etc., but item 6 of Lemma 7.18 guarantees that this cannot continue infinitely often, so, starting from some j , all S_j will contain the same path sequent Π' that prefix-subsumes Π . Thus, we proved:

- (7.4) *for every path sequent Π in any S_i there exists a $\Pi' \in S_\infty$ such that Π' prefix-subsumes Π .*

Moreover, using similar considerations one can strengthen this result to

- (7.5) *for every path sequent Π in any S_i there exists a $\Pi' \in S_\infty$ such that Π' prefix-subsumes Π , and if $\Pi'' \in S_\infty$ prefix-subsumes Π' , then $\Pi'' = \Pi'$.*

For any path sequent Π (not necessarily in some S_i) we call a *trace* of Π in \mathcal{S} any $\Pi' \in S_\infty$ such that Π' prefix-subsumes Π and no $\Pi'' \in S_\infty$ prefix-subsumes Π' . Now (7.5) can be reformulated: every $\Pi \in S_i$ has a trace in \mathcal{S} .

Note that (7.3) obviously follows from a slightly more general statement

- (7.6) *Every path sequent Π that appears in a fair derivation \mathcal{T} in $\{\mathbb{P}_{inv}^{G, \succ}\}$ has a trace in \mathcal{S} .*

We will now prove (7.6), and then we are done. Let $\mathcal{T} = T_0 \triangleright T_1 \triangleright \dots$. Take the least i such that $\Pi \in T_i$. (7.6) will be proved by induction on i . If $i = 0$, then Π is an axiom, therefore it belongs to S_0 , and by (7.5) has a trace in \mathcal{S} . If $i > 0$, then Π is obtained by an inference I of $\mathbb{P}_{inv}^{G, \succ}$ from path sequents in T_{i-1} . By the induction

hypothesis, these path sequents have traces in \mathcal{S} . We use the case analysis on the inference I , but before we will note some general properties related to redundancies and prefix-subsumption. Namely, we prove the following two statements.

(7.7) *Suppose Π is a path sequent that contains a pair of conflicting paths π_1, π_2 and Π prefix-subsumes Π' . Then Π' contains a pair of conflicting paths, too.*

(7.8) *Suppose $\pi \succ \Pi'$ and Π prefix-subsumes Π' . Then $\pi \succ \Pi$.*

(7.7) follows from the following observation: if $\pi_1, \pi_2 \in \Pi$ and Π prefix-subsumes Π' , then there exist $\pi'_1, \pi'_2 \in \Pi'$ such that $\pi_1 \sqsubseteq \pi'_1$ and $\pi_2 \sqsubseteq \pi'_2$. So if π_1 being in conflict with π_2 implies π'_1, π'_2 are conflicting, too. To prove (7.8), note the following: Π consists of prefixes of Π' , and every path is greater than any its prefix with respect to \succ , so π is greater than any path sequent in Π .

Let us now return to the proof of (7.6).

1. The inference I has the form

$$\frac{\Pi, \pi \wedge_I}{\Pi, \pi} (\wedge_I).$$

Let Γ be a trace of $\Pi, \pi \wedge_I$ in \mathcal{S} . Consider three possible cases:

- (a) *Case Γ prefix-subsumes Π .* Then Γ also prefix-subsumes Π, π , and is a trace of Π, π .
- (b) *Case Γ has the form $\Gamma', \pi \wedge_I$, where Γ' prefix-subsumes Π .* Consider the inference

$$\frac{\Gamma', \pi \wedge_I}{\Gamma', \pi} (\wedge_I). \quad (7.9)$$

Let us show that this inference is not redundant in $\mathbb{P}_{inv}^{G, \succ}$. Evidently, Γ', π contains no pair of paths of different modal lengths, since so is $\Gamma', \pi \wedge_I$. (7.7) guarantees that Γ', π contains no conflicting pairs of paths, because so is Π, π , and finally, (7.8) guarantees that $\pi \succ \Gamma'$, because $\pi \succ \Pi$.

Since inference (7.9) is nonredundant and \mathcal{S} is fair, \mathcal{S} contains this inference. By (7.5), \mathcal{S} contains a trace of Γ', π ; evidently, this trace is also a trace of Π, π .

Below we will no more prove that the inferences in \mathcal{S} are nonredundant, since the proof is similar to the one in this case.

- (c) *Case Γ has the form Γ', π' , where Γ' prefix-subsumes Π and $\pi' \sqsubset \pi \wedge_I$.* Note that if $\pi' \sqsubset \pi$, then Γ', π' also prefix-subsumes Π, π . So we only consider the case $\pi' = \pi$. Using the definition of prefix-subsumption, one can show that either Γ', π prefix-subsumes Π , in which case we are done, or $\pi \in \Gamma'$. Let us show that $\pi \in \Gamma'$ is impossible. Indeed, $\pi \in \Gamma'$ implies that Γ, π has a form Γ'', π, π for some Γ'' . Since \mathcal{S} is fair, the contraction rule should be applied in \mathcal{S} to Γ'', π, π , obtaining Γ'', π which prefix-subsumes Γ'', π, π . By (7.5) Γ'', π has a trace in \mathcal{S} , then this trace prefix-subsumes Γ'', π, π . But Γ'', π, π is a trace itself, and cannot be prefix-subsumed by any other path sequent in \mathcal{S}_∞ . Contradiction.

2. The inference I has the form

$$\frac{\Pi_1, \pi \vee_l \quad \Pi_2, \pi \vee_r}{\Pi_1, \Pi_2, \pi} (\vee).$$

Take traces Γ_1 and Γ_2 of $\Pi_1, \pi \vee_l$ and $\Pi_2, \pi \vee_r$, respectively, in \mathcal{S} . Similar to the previous case, we consider the following subcases:

- (a) *Case (either Γ_1 prefix-subsumes Π_1 or Γ_2 prefix-subsumes Π_2).* Then either Γ_1 or Γ_2 prefix-subsumes Π_1, Π_2, π .
- (b) *Case Γ_1 contains $\pi \vee_l$ and Γ_2 contains $\pi \vee_r$.* Then Γ_1 has the form $\Gamma'_1, \pi \vee_l$ and Γ_2 has the form $\Gamma'_2, \pi \vee_r$ such that Γ_1 and Γ_2 prefix-subsume Π_1 and Π_2 , respectively. Since \mathcal{S} is fair, it contains an inference

$$\frac{\Gamma'_1, \pi \vee_l \quad \Gamma'_2, \pi \vee_r}{\Gamma'_1, \Gamma'_2, \pi} (\vee).$$

By (7.5), \mathcal{S} contains a trace of $\Gamma'_1, \Gamma'_2, \pi$; evidently, this trace is also a trace of Π_1, Π_2, π .

- (c) *Case (Γ_1 has the form Γ'_1, π' , where Γ'_1 prefix-subsumes Π_1 and $\pi' \sqsubset \pi \vee_l$).* This case is similar to the respective case for the conjunction above.
- (d) *Case (Γ_2 has the form Γ'_2, π' , where Γ'_2 prefix-subsumes Π_2 and $\pi' \sqsubset \pi \vee_r$).* This case is symmetric to the previous one.

Other cases are considered in [Voronkov 2001].

The proof is complete. □

However, the completeness result for prefix-subsumption has some applications besides the possibility of removing subsumed sequents. Let us look at some interesting observations that come out of this result. Consider any (\wedge_*) -inference:

$$\frac{\Pi, \pi \wedge_*}{\Pi, \pi} (\wedge_*).$$

Note that the conclusion of this inference prefix-subsumes the premise. Thus, the premise can be discarded as soon as the rule is applied. Even more, when $\pi \wedge_*$ is not maximal in $\Pi, \pi \wedge_*$, so that the inference is redundant, it makes sense to apply this inference, because the premise can be discarded after. If we apply (\wedge_*) -rules in an eager way, as soon as they are applicable, any path $\pi \wedge_*$ becomes “short-lived.” This observation has a far-reaching consequence: one can modify the calculus $\mathcal{P}_{inv}^{G, \succ}$ so that such paths do not occur in sequents at all. This modification is similar to the optimization used for the named calculus in Section 5.3. Such an optimization first appeared for classical logic in [Voronkov 1985], and then in a more accessible paper [Voronkov 1990b]. It has also been used in tableau calculi in [Degtyarev and Voronkov 1996b] as a *least conjunctive superformula* optimization. For logic **K** it has been implemented in the system KX [Voronkov 2000a].

Another interesting consequence of the use of prefix-subsumption is the following. Suppose that during search for a refutation we obtained a path sequent $\Delta = \Pi, \pi, \pi'$

$$\begin{array}{c}
\frac{}{\Gamma, A, \neg A} (Ax) \qquad \frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \vee B} (\vee) \qquad \frac{\Gamma, A, B}{\Gamma, A \wedge B} (\wedge) \\
\\
\frac{\Gamma, A(y)}{\Gamma, \exists x A(x)} (\exists) \qquad \frac{\Gamma, A(t)}{\Gamma, \forall x A(x)} (\forall)
\end{array}$$

The rule (\exists) satisfies the eigenvariable condition: y is a variable not occurring in Γ .

Figure 30: Calculus \mathbf{G}_{tab} for Grishin's logic

such that $\pi \sqsubseteq \pi'$. Consider the path sequent $\Gamma = \Pi, \pi', \pi'$. This sequent is prefix-subsumed by Δ , so it can be added to the search space without compromising soundness or completeness. But if we add Γ to the search space, by contraction we can derive Π, π' that subsumes Δ . Therefore Π, π, π' can be replaced by a simpler sequent Π, π' , as soon as π is a prefix of π' .

8. Logics without the contraction rules

The use of the inverse method for proving the decidability of various logics has a long history. Gentzen [1934] proved the decidability of propositional intuitionistic logic essentially using the inverse method technique. Maslov [1966] used the inverse method to prove the decidability of several fragments of first-order classical logic.

In this section we demonstrate a nice application of the inverse method for the decidability of some **contraction-free logics**: logics without the contraction rule. A number of such logics have been proposed. For example Grishin [1981] introduced such a logic with the purpose of providing alternative foundations for logic and set theory, while Ketonen and Weyhrauch [1984] introduced a contraction-free fragment of predicate calculus with the aim of defining “simply provable” formulas. Another example of such a logic is the linear fragment of linear logic [Girard, Lafont and Taylor 1989].

Let us show that the predicate version of Grishin's [1981] logic is decidable.⁸ The decidability of this logic was proved by the inverse method in [Voronkov 1992]. The technique used in the proof is very general and can be used to prove the decidability of all other contraction-free logics mentioned here. The standard translation to negation normal form is also valid for Grishin's logic, so we present the logic for NNFs.

A sequent calculus for the predicate version of Grishin's logic, denoted \mathbf{G}_{tab} is shown in Figure 30. Compare this calculus with the calculus \mathbf{C}_{tab} for NNFs, see Figure 10 on page 205. The only difference between the two calculi is the rule (\forall) , in which the quantified formula $\forall x A(x)$ is not duplicated.

⁸Grishin writes that his logic is decidable but gives no proof of it.

The main aim of this section is to design an inverse calculus for Grishin's logic and prove the decidability of this logic using proof-theoretic properties of the calculus. With the technique developed so far, it is almost a mechanical exercise. For aesthetic reasons, we will show how to mix ingredients used so far in a slightly new way. For example, we use paths instead of names and make a first-order calculus that is technically simpler than those used previously by using a notion of variable sequence on a path.

The proof of decidability of Grishin's logic is based on the following ideas, developed in the rest of this section

- Design a path calculus corresponding to the sequent calculus of G_{tab} . Prove a Bisimulation Lemma for this calculus, thus obtaining completeness.
- Design a corresponding inverse path calculus. This time we will skip the construction of a ground inverse method calculus and build a free-variable calculus immediately. In the completeness proof for this calculus, we will merge the Subsequent Lemma and Lifting. Then we will use proof-theoretic properties of the inverse path calculus to prove the decidability.

8.1. Tableau path calculus and bisimulation lemma

It will be convenient for us to deal with occurrences of subformulas in the goal formula instead of subformulas themselves. We already know two ways of formalizing the notion of occurrence: name calculi and path calculi. Name calculi were introduced for predicate logics in Section 5.1, path calculi were introduced for propositional logic in Section 7. As one can see, path calculi are technically more convenient since paths contain a lot of information about the occurrence of a subformula. In this section we will extend path calculi to the first-order case, using the technique of designing first-order inverse method calculi.

8.1. DEFINITION (path in the first-order case). A *path* is any finite sequence of symbols $\wedge_l, \wedge_r, \vee_l, \vee_r, \forall, \exists$. Let G be a formula. The notion of *path in G* , and the *subformula of G on a path π* , are defined as in Definition 7.2 on page 234, but with the following additional cases.

Suppose that π is a G -path and $G|_{\pi} = F$. Then

- If F has the form $\forall x F_1$, then $\pi\forall$ is a G -path, and $G|_{\pi\forall} = F_1$. In this case we call π a *\forall -path*.
- If F has the form $\exists x F_1$, then $\pi\exists$ is a G -path, and $G|_{\pi\exists} = F_1$. In this case we call π a *\exists -path*.

In the propositional case, each occurrence of a subformula in the goal formula G can be uniquely identified by its path. In the first-order case, this is true for free subformulas, but not for arbitrary subformulas. Any subformula of G can be obtained from a free subformula by applying a substitution, so any subformula F of G can be identified by a pair π, σ such that $F = G|_{\pi}\sigma$. This would give us a formalization similar to those used for designing free-variable calculi for classical

$$\begin{array}{c}
\frac{}{\Gamma, \pi_1(\bar{s}), \pi_2(\bar{t})} (Ax) \\
\\
\frac{\Gamma, \pi \vee_l(\bar{s}) \quad \Gamma, \pi \vee_r(\bar{s})}{\Gamma, \pi(\bar{s})} (\vee) \qquad \frac{\Gamma, \pi \wedge_l(\bar{s}), \pi \wedge_r(\bar{s})}{\Gamma, \pi(\bar{s})} (\wedge) \\
\\
\frac{\Gamma, \pi \exists(\bar{s}, y)}{\Gamma, \pi(\bar{s})} (\exists) \qquad \frac{\Gamma, \pi \forall(\bar{s}, t)}{\Gamma, \pi(\bar{s})} (\forall)
\end{array}$$

The rule (\exists) satisfies the eigenvariable condition: y is a variable not occurring in Γ or \bar{s} . In the rule (Ax) we have $G|_{\pi_1(\bar{s})} = \neg G|_{\pi_2(\bar{t})}$.

Figure 31: Tableau path calculus for Grishin's logic

logic in Section 3.3 and intuitionistic logic in Section 4.1. However, we do not want to work with such pairs and will try a simpler presentation in the spirit of a name calculus. We will use paths as names and use expressions like $\pi(t_1, \dots, t_n)$, where π is a path, to encode arbitrary subformulas of the goal formula.

8.2. DEFINITION (variable sequence and path name). The notion of *variable sequence on a G -path* π , denoted $\text{var}(G, \pi)$, is defined as follows

1. $\text{var}(G, \epsilon)$ is the empty sequence of variables.
2. If $\pi \mathbb{X}_*$ is a path in G , then $\text{var}(G, \pi \mathbb{X}_*) = \text{var}(G, \pi)$.
3. If $\text{var}(G, \pi)$ is a sequence (x_1, \dots, x_n) and $G|_\pi$ has the form $\forall x F$ (respectively $\exists x F$), then $\text{var}(G, \pi \forall) = (x_1, \dots, x_n, x)$ (respectively, $\text{var}(G, \pi \exists) = (x_1, \dots, x_n, x)$).

Let π be a path in G , $\text{var}(G, \pi) = (x_1, \dots, x_n)$ and t_1, \dots, t_n be a sequence of terms. Then we call the expression $\pi(t_1, \dots, t_n)$ the *G -path name* of the formula $G|_\pi\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$.

Consider, for example, the formula $G = \forall x(\exists y P(x, y) \vee \forall z(\neg P(x, z) \vee Q(z)))$. Then $\pi = \forall \vee_r \forall \vee_r$ is a path in G , and we have $G|_\pi = Q(z)$ and $\text{var}(G, \pi) = (x, z)$. Further, $\forall \vee_r \forall \vee_r (f(a), b)$ is a G -path name of the subformula $Q(b)$. In the sequel we will simply write *path name*, since G will always be clear from the context. If $\pi(\bar{t})$ is a G -path name for a subformula F of G , we will use notation $G|_{\pi(\bar{t})}$ to denote F . Note that this notation is consistent with the one with used for propositional **K**.

It is easy to see that $\text{var}(G, \pi)$ contains all variables in $\text{free}(G|_\pi) \setminus \text{free}(G)$. In particular, if G is closed, then $\text{var}(G, \pi)$ contains all free variables of $G|_\pi$.

The *tableau path calculus for Grishin's logic* is shown in Figure 31. Its sequents are multisets of G -path names.

Similar to the case of logic **K**, we define the *formula image* of a path sequent $\pi_1(\bar{t}_1), \dots, \pi_n(\bar{t}_n)$ as the sequent $G|_{\pi_1(\bar{t}_1)}, \dots, G|_{\pi_n(\bar{t}_n)}$.

Using the same considerations as in Bisimulation Lemma 7.3 on page 235, we can prove

8.3. LEMMA (Bisimulation Lemma for Grishin's logic). *(i) Let D be a derivation in the tableau path calculus for G . Then the formula image of D is a derivation of G in \mathbf{G}_{tab} . (ii) Let D' be a derivation of a sequent A_1, \dots, A_n in \mathbf{G}_{tab} , and $\pi_1(\bar{t}_1), \dots, \pi_n(\bar{t}_n)$ be G -path names of A_1, \dots, A_n . Then there exists a derivation D of $\pi_1(\bar{t}_1), \dots, \pi_n(\bar{t}_n)$ in the tableau path calculus for G such that D' is the formula image of D . (iii) Both (i) and (ii) also hold if "derivation" is replaced by "refutation" everywhere.* \square

This lemma implies soundness and completeness of the tableau path calculus for G .

8.4. THEOREM (Completeness). *Let G be a closed formula. Then G has a refutation in \mathbf{G}_{tab} if and only if ϵ has a refutation in the tableau path calculus for G .* \square

8.2. Inverse path calculus and decidability

The construction of a free-variable inverse path calculus for Grishin's logic is now straightforward. We suggest the reader try constructing it before looking at Figure 32. There are two tricky rules: \vee and \wedge . The rule \vee in the tableau path calculus has a hidden contraction rule on Γ , which we should take care of in the inverse calculus. The rule (\wedge) now gets split into three rules. We need three rules since we do not have the contraction rule any more, and the construction used in the proof of the Subsequent Lemma on page 190 may not work.

Subsequent Lemma and Lifting will now be merged using the following notion. We say that a path sequent Π_1 **subsumes** a path sequent Π_2 if there exists a substitution σ such that $\Pi_1\sigma$ is a submultiset of Π_2 .

8.5. LEMMA (Subsumption Lemma). *Let Π_2 have a refutation in the tableau path calculus for Grishin's logic. Then there exists a path sequent Π_1 such that (i) Π_1 subsumes Π_2 and (ii) Π_1 has a refutation in the free-variable inverse path calculus for Grishin's logic.* \square

As usual, we obtain soundness and completeness as a consequence of this lemma.

8.6. THEOREM (Completeness). *Let G be a closed formula. Then G has a refutation in \mathbf{G}_{tab} if and only if ϵ has a refutation in the free-variable inverse path calculus for G .* \square

We are just one step from proving the decidability of Grishin's logic. Let us note one useful property of the free-variable inverse path calculus. We call a path sequent **prefix-free** if for every two different occurrences of path names $\pi_1(\bar{s}), \pi_2(\bar{t})$ in Π , π_1 is not a prefix of π_2 and π_2 is not a prefix of π_1 .

$$\begin{array}{c}
\frac{}{\Gamma, \pi_1(\bar{s}), \pi_2(\bar{t})} (Ax) \quad \frac{\Gamma_1, \Delta_1, \pi \vee_l(\bar{s}) \quad \Gamma_2, \Delta_2, \pi \vee_r(\bar{t})}{(\Gamma_1, \Delta_1, \Delta_2, \pi(\bar{s}))\sigma} (\vee) \\
\frac{\Gamma, \pi \wedge_l(\bar{s})}{\Gamma, \pi(\bar{s})} (\wedge_l) \quad \frac{\Gamma, \pi \wedge_r(\bar{t})}{\Gamma, \pi(\bar{t})} (\wedge_r) \quad \frac{\Gamma, \pi \wedge_l(\bar{s}), \pi \wedge_r(\bar{t})}{(\Gamma, \pi(\bar{s}))\text{mgu}(\bar{s}, \bar{t})} (\wedge) \\
\frac{\Gamma, \pi \exists(\bar{s}, y)}{\Gamma, \pi(\bar{s})} (\exists) \quad \frac{\Gamma, \pi \forall(\bar{s}, t)}{\Gamma, \pi(\bar{s})} (\forall)
\end{array}$$

For the rule (Ax) the following conditions holds: there exists free subformulas $\neg A$ and B of G and a most general unifier σ of A and B such that $\pi_1(\bar{s})$ is a path name for $\neg A\sigma$ and $\pi_2(\bar{t})$ is a path name for $B\sigma$.

In the rule (\vee) σ is a most general unifier of Γ_1, \bar{s} and Γ_2, \bar{t} .

The rule (\exists) satisfies the eigenvariable condition: y is a variable not occurring in Γ or \bar{s} .

Figure 32: Free-variable inverse path calculus for Grishin's logic

8.7. LEMMA. *Let D be a refutation of ϵ in the free-variable inverse path calculus and Π be a path sequent in D . Then Π is prefix-free.*

PROOF. Suppose, by contradiction, that some path sequent Π in D is not prefix-free. By inspecting the rules of the free-variable inverse path calculus one can see that any path sequent below Π in D is also not prefix-free. Therefore, the bottom sequent of the derivation must contain at least two path names, so it cannot be ϵ . \square

The decidability of Grishin's logic will be proved as follows. We will build a partial order $>$ on path sequents which satisfies the following properties:

1. for every inference in the free-variable path calculus the conclusion of the inference is strictly greater than each premise w.r.t. $>$;
2. in every infinite increasing w.r.t. $>$ chains of path sequents, there is a path sequent that is not prefix-free, and therefore cannot occur in a refutation of ϵ .

These properties will immediately imply the decidability of Grishin's logic.

First, we construct such a partial order $>$ for multisets of paths. For two multisets of paths Π_1 and Π_2 we have $\Pi_1 > \Pi_2$ if Π_1 can be obtained from Π_2 by a finite sequence of the following steps:

- addition of any path;
- replacement of any path by its prefix.

For example, we have $\forall, \exists \wedge_l > \forall \vee_r$ because we can obtain $\forall, \exists \wedge_l$ from $\forall \vee_r$ in two steps:

- addition of the path $\exists \wedge_l$;

- replacement of the path $\forall V_r$ by its prefix \forall .

It is not hard to argue that $>$ is a partial order.

For path sequents we define $\pi_1(\bar{s}_1), \dots, \pi_n(\bar{s}_n) > \pi'_1(\bar{t}_1), \dots, \pi'_m(\bar{t}_m)$ if and only if $\pi_1, \dots, \pi_n > \pi'_1, \dots, \pi'_m$.

Let us prove the above two properties. When we prove them, we will freely shift from path names $\pi(\bar{t})$ to paths π and back.

1. *For every inference in the free-variable path calculus the conclusion of the inference is strictly greater than each premise w.r.t. $>$.* This property trivially holds for every inference of the calculus. For example, in the rule (V) the conclusion can be obtained from the left premise by adding all paths in Δ_2 and replacement of πV_l by its prefix π .
2. *In every infinite increasing w.r.t. $>$ chains of path sequents, there is a path sequent that is not prefix-free, and therefore cannot occur in a refutation of ϵ .* Suppose that we have an infinite increasing chain of paths sequents

$$\dots \Pi_2 > \Pi_1 > \Pi_0.$$

Note that for any path sequent Π_i , we can replace a path by its prefix only a finite number of times. This means that the number of path names in Π_i 's is not bound by any finite number. Since the number of G -paths is finite, this means that some Π_i (with a large enough number of elements) contains two different occurrences of the same path.

Note that this observation can be generalized to

8.8. LEMMA. *The depth of derivations of ϵ is bound by the number $d(G) = m \cdot n$, where m is the length of the longest G -path and n is the number of all G -paths.*

8.9. THEOREM. *Grishin's logic is decidable.*

PROOF. By Completeness Theorem 8.6, it is enough to show how to decide whether ϵ has a refutation in the free-variable inverse path calculus for G . By the above observation, the depth of any refutation of ϵ is bounded by a finite number $d(G)$ that can be effectively computed for G . But we know that the free-variable inverse path calculus has the finite rule property, so we can effectively generate all derivations up to this depth. \square

9. Conclusion

9.1. History

In this section we briefly overview the history of the inverse method. We do not mention *all* publications on the inverse method here, we also do not mention who was the first to treat a particular nonclassical logic. Rather, we only consider papers that introduce some novelties in the technique, for example free-variable calculi.

The first paper that discussed the inverse method was [Gentzen 1934]. Gentzen proposed a decision procedure for intuitionistic propositional logic, as an application of cut-free sequent calculi. Let us quote the relevant part of the paper.

On the basis of the *Hauptsatz* we can state a simple procedure for deciding of a formula of propositional logic — i.e., a formula without object variables — whether or not it is classically or intuitionistically true. (For classical propositional logic a simple solution has actually been known for some time, cf., e.g. p.11 of Hilbert-Ackermann.)

First we prove the following *lemma*:

A sequent in whose antecedent one and the same formula does not occur more than three times as an *S*-formula, and in whose succedent, furthermore, one and the same formula does not occur more than three times as an *S*-formula, will be called a “reduced sequent”. The following lemma now holds:

Every *LJ*- or *LK*-derivation whose endsequent is reduced, may be transformed into an *LJ*- or *LK*-derivation with the same endsequent, in which all sequents are reduced (and in which no cuts occur if the original derivation did not contain any).

Consider now a sequent not containing an object variable. We wish to decide whether or not it is intuitionistically or classically true. We can begin by taking in its place an equivalent reduced sequent $\mathfrak{S}q$.

The number of all reduced sequents whose *S*-formulae are subformulae of the *S*-formulae of $\mathfrak{S}q$ is obviously finite. The decision procedure may therefore be carried out without further complications in the following way:

We consider the finite system of sequents in question and investigate first of all, which of these sequents are basic sequents⁹. Then we examine each of the remaining sequents to determine whether there occurs an inference figure in which the sequent in question is the lower sequent and in which there occur as upper sequents one or two of those sequents that have already been found to be derivable. If this is the case, the sequent is added to the derivable sequents. (All this is obviously decidable.) We continue in this way until either the sequent $\mathfrak{S}q$ itself turns out to be derivable, or until the procedure yields no new derivable sequents. In the latter case the sequent $\mathfrak{S}q$ is not derivable at all in the calculus under consideration (*LJ* or *LK*). We have therefore succeeded in establishing the validity of that sequent.

Our decision procedure can be formulated in a way better suited to the needs of practical application; yet the above presentation was intended only to indicate a possibility in principle.

The reason for the introduction of “reduced sequents” was due to the fact that in Gentzen’s original system sequents were sequences of formulas and he had several structure rules for handling such sequents.

The term “inverse method” and an inverse method calculus for classical logic were introduced in Maslov [1964]. The calculus was defined for classical first-order logic without equality and function symbols, however it was noted that

“it is a relatively simple matter to develop the inverse method to include the case of the classical predicate calculus with constant functional symbols”

⁹i.e. axioms.

Skolemization was not used, and the method was applied to formulas of the form

$$Q_1 x_1 \dots Q_n x_n \left(\bigvee_{i=1}^{\mu} \bigwedge_{j=1}^{\delta_i} D_{i,j} \right).$$

where $Q_1 x_1 \dots Q_n x_n$ is a quantifier prefix and $D_{i,j}$ are disjunctions of literals. Maslov [1964] did not refer to sequent calculi and the subformula property, and the method was presented as based on Herbrand's theorem. It was emphasized as well that the notion of metavariables (corresponding to free variables in the modern terminology) proposed by Shanin in 1962 served as the author's point of departure.

Maslov [1964] showed how to apply the method for establishing the decidability of classes of formulas, in particular the class of formulas of the form

$$\forall^* \exists^* \forall^* \left(\bigvee_{i=1}^{\mu} \bigwedge_{j=1}^{\delta_i} D_{i,j} \right).$$

where $\delta_1 \leq 2, \dots, \delta_{\mu} \leq 2$ (called the Maslov class in [Börger, Grädel and Gurevich 1997]).

Maslov [1966] applied the inverse method to prove in a uniform way the *decidability of several classes of formulas* of the first-order classical predicate calculus, including some new classes. Maslov [1967] modified the inverse method to *arbitrary formulas with function symbols*. Relationships between the *inverse method* and (*hyper*)*resolution* were noted by [Maslov 1969, Maslov 1971b, Kuechner 1971]. The first implementation of the inverse method for classical logic is due to Davydov, Maslov, Mints, Orevkov and Slissenko [1969]. Maslov [1971a] showed how to generalize the inverse method to *logic with equality*, but not in the form of a free-variable calculus¹⁰. Mints [1981] (published in Russian) was the first to apply the inverse method (under the name of resolution) to *nonclassical* propositional logics. This and other early papers on the inverse method for nonclassical logics are overviewed in [Mints 1990]. A *free-variable* inverse calculus for classical logic without equality was introduced in [Voronkov 1985], a more accessible reference is [Voronkov 1990b]. Inverse calculi for *nonclassical predicate logics* appeared in [Voronkov 1992], already with free variables. This paper also proposed a uniform technique for proving redundancy criteria. A *free-variable method* for classical logic *with equality* was introduced in [Degtyarev and Voronkov 1995a] with a number of redundancy criteria, for example those based on reduction orderings and the basic strategy¹¹. A first implementation of the inverse method for predicate intuitionistic logic is described in [Tammet 1996]. *Path calculi* were introduced in [Voronkov 2000b].

¹⁰Maslov [1971a] formulated completeness of the inverse method in terms of existence of a substitution with some properties, so his formulation were halfway to free-variable methods, since no algorithm for finding such a substitution has been proposed. In fact, the problem of the existence of such a substitution for logic with equality is undecidable, since one can reduce to it simultaneous rigid *E*-unification [Degtyarev and Voronkov 1996f].

¹¹Free-variable methods for some nonclassical logics with equality do not exist, in a certain sense, for details see [Voronkov 1998c].

9.2. *Limits of the technique and future research*

9.2.1. *More expressive description logics*

It seems that the inverse method is applicable to logics having a suitable subformula property, including logics with the analytic cut rule. However, there is a long way to go in applying it to more expressive logics, for example, some description logics covered in [Calvanese, Giacomo, Lenzerini and Nardi 2001] (Chapter 23 of this Handbook). Such logics are usually defined by a possible world semantics or by tableau systems with explicit notation for worlds. It is unclear whether such tableau systems can be easily made into inverse calculi: this requires further research.

9.2.2. *New completeness proofs*

Moreover, proving completeness of inverse calculi using the detour through tableau calculi may be unnecessary, provided that new methods to prove completeness are developed. Development of such methods is a very interesting research topic. If such methods are developed, it is possible that completeness proofs for calculi with redundancy criteria will be considerably simplified.

9.2.3. *Implementation*

It would be interesting to implement the inverse method for nonclassical propositional logics efficiently, so that it can compete with the best available systems, for example FaCT, DLP [Horrocks and Patel-Schneider 1999] or *SAT [Giunchiglia, Giunchiglia and Tacchella 2000]. The current implementation of the system KY based on the inverse method [Voronkov 2000a] is still not as fast as these systems.

For first-order intuitionistic logic the inverse method was implemented efficiently by Tammet [1996], but the implementations of first-order nonclassical logics are not so advanced as the implementations of propositional logics, and especially description logics.

9.2.4. *Bidirectional systems*

It would be interesting to develop bidirectional theorem proving systems, where proof-search will be carried out by a combination of top-down (tableaux) and bottom-up (inverse) proof-search methods. This requires new methods of proving completeness of calculi with redundancy criteria, in order to ensure that the criteria developed for the tableau part and the inverse part are compatible.

9.3. *Warning*

If you had enough patience to read this chapter to the very end, you become very experienced in cooking. You can take almost any logic (well, any logic complying with the Subformula Property), and make a meal out of it. The aim of this subsection is to warn you (see also the Fast Food Warning in Section 5.7): with prolific

food you will become fat and your friends and colleagues will despise you¹². The calculi for the inverse method can be obtained in an essentially automatic way for most logics with the subformula property. To obtain *more elegant* calculi by using variations on the initial calculi or by using the completeness proofs directly is an appealing task. To obtain inverse method calculi with *new unusual features* is of a definite interest. To find *new redundancy criteria* and prove a combination of them complete is a very creative task. To *implement* the inverse method for propositional nonclassical logics *with efficiency comparable with that of the tableau method* is a challenge. To design a *bidirectional system with redundancy criteria* is one of the most important problems in this area. The design and implementation of such systems can become the *major research area for logics used in applications*, for example description logics.

Acknowledgments

We thank the second readers whose remarks have improved the article considerably. However, the authors are the only responsible for all inaccuracies that can be present.¹³

Bibliography

- ANDREWS P. [2001], Classical type theory, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 15, pp. 965–1007.
- APT K. [1990], Logic programming, in J. Van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B: Formal Methods and Semantics, Elsevier Science, Amsterdam, chapter 10, pp. 493–574.
- AUFFRAY I., ENJALBERT P. AND HEBRARD J.-J. [1990], 'Strategies for modal resolution: Results and problems', *Journal of Automated Reasoning* 6, 1–38.
- BAADER F. AND SNYDER W. [2001], Unification theory, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 8, pp. 445–532.
- BAAZ M., EGLY U. AND LEITSCH A. [2001], Normal form transformations, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 5, pp. 273–333.
- BAAZ M., FERMÜLLER C. AND SALZER G. [2001], Automated deduction for many-valued logics, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 20, pp. 1355–1402.
- BACHMAIR L. AND GANZINGER H. [2001], Resolution theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19–99.

¹²There was a lot of activity in Russia in the area of Lyapunov's vector functions method. In the 1970s Valeri Matrosov with his group implemented a system that proved theorems in this area in a completely automatic way. It has been shown that many articles published in respected scientific journals could have been written by a computer. Yuri Ershov characterized the main achievement of Matrosov thus: they proved that a part of mathematics has no right to exist. It is definitely *not* the aim of this chapter to seize the research on the inverse method.

¹³We would like to mention that the second second reader has a different viewpoint on the history of the inverse method, see e.g., [Maslov, Mints and Orevkov 1983].

- BARENDREGT H. AND GEUVERS H. [2001], Proof-assistants using dependent type systems, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 18, pp. 1149–1238.
- BETH E. W. [1956], 'Semantic construction of intuitionistic logic', *Noord-Hollandsche Uitgevers Maatschappij*.
- BOCKMAYR A. AND WEISPFENNING V. [2001], Solving numerical constraints, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 12, pp. 749–842.
- BÖRGER E., GRÄDEL E. AND GUREVICH Y. [1997], *The Classical Decision Problem*, Springer Verlag.
- BOWEN K. [1979], *Model Theory for Modal Logic*, Vol. 127 of *Synthese Library*, D. Reidel.
- BOY DE LA TOUR T. [1990], Minimizing the number of clauses by renaming, in M. Stickel, ed., 'Proc. 10th CADE', Vol. 449 of *Lecture Notes in Artificial Intelligence*, pp. 558–572.
- BUNDY A. [2001], The automation of proof by mathematical induction, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 13, pp. 845–911.
- CALVANESE D., GIACOMO G. D., LENZERINI M. AND NARDI D. [2001], Reasoning in expressive description logics, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 23, pp. 1581–1634.
- CHAGROV A. AND ZAKHARYASCHEV M. [1996], *Modal Logic*, Oxford University Press.
- CHOU S. C. AND GAO X. S. [2001], Automated reasoning in geometry, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 11, pp. 705–747.
- CLARKE E. AND SCHLINGLOFF H. [2001], Model checking, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 24, pp. 1635–1790.
- COMON H. [2001], Inductionless induction, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 14, pp. 913–962.
- CURRY H. [1963], *Foundations of Mathematical Logic*, McGraw-Hill, New York.
- DAVIS M. [2001], The early history of automated deduction, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 1, pp. 3–15.
- DAVYDOV V., MASLOV S., MINTS G., OREVKOV V. AND SLISSENKO A. [1969], 'A computer algorithm of establishing deducibility based on the inverse method', *Zapiski Nauchnykh Seminarov LOMI* 16, 8–19.
- DE NIVELLE H., SCHMIDT R. AND HUSTADT U. [2000], 'Resolution-based methods for modal logics', *Logic Journal of the IGPL* 8(3), 265–292.
- DEGTYAREV A. AND VORONKOV A. [1994a], Equality elimination for semantic tableaux, UPMail Technical Report 90, Uppsala University, Computing Science Department.
- DEGTYAREV A. AND VORONKOV A. [1994b], Equality elimination for the inverse method and extension procedures, UPMail Technical Report 92, Uppsala University, Computing Science Department.
- DEGTYAREV A. AND VORONKOV A. [1995], Equality elimination for the inverse method and extension procedures, in C. Mellish, ed., 'Proc. International Joint Conference on Artificial Intelligence (IJCAI)', Vol. 1, Montréal, pp. 342–347.
- DEGTYAREV A. AND VORONKOV A. [1996a], Equality elimination for the tableau method, in J. Calmet and C. Limongelli, eds, 'Design and Implementation of Symbolic Computation Systems. International Symposium, DISCO'96', Vol. 1128 of *Lecture Notes in Computer Science*, Karlsruhe, Germany, pp. 46–60.
- DEGTYAREV A. AND VORONKOV A. [1996b], 'The undecidability of simultaneous rigid E -unification', *Theoretical Computer Science* 166(1–2), 291–300.
- DEGTYAREV A. AND VORONKOV A. [1996c], What you always wanted to know about rigid E -unification, in J. Alferes, L. Pereira and E. Orłowska, eds, 'Logics in Artificial Intelligence.

- European Workshop, JELIA'96', Vol. 1126 of *Lecture Notes in Artificial Intelligence*, Évora, Portugal, pp. 50–69.
- DEGTYAREV A. AND VORONKOV A. [2001a], Equality reasoning in sequent-based calculi, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 10, pp. 609–704.
- DEGTYAREV A. AND VORONKOV A. [2001b], The inverse method, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 4, pp. 179–272.
- DESHOWITZ N. AND PLAISTED D. [2001], Rewriting, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 9, pp. 533–608.
- DIX J., FURBACH U. AND NIEMELÄ I. [2001], Nonmonotonic reasoning: Towards efficient calculi and implementations, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 19, pp. 1241–1354.
- DOWEK G. [2001], Higher-order unification and matching, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 16, pp. 1009–1062.
- DYCKHOFF R. [1992], 'Contraction-free sequent calculi for intuitionistic logic', *Journal of Symbolic Logic* 57(3), 795–807.
- FERMÜLLER C., LEITSCH A., HUSTADT U. AND TAMMET T. [2001], Resolution decision procedures, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 25, pp. 1791–1849.
- FITTING M. [1983], *Proof methods for modal and intuitionistic logics*, Vol. 169 of *Synthese Library*, Reidel Publ. Comp.
- FITTING M. [1996], *First Order Logic and Automated Theorem Proving*, Springer Verlag, New York. Second edition.
- FITTING M. AND MENDELSON R. [1998], *First-Order Modal Logic*, Kluwer Academic Publishers.
- FITTING M., THALMANN L. AND VORONKOV A. [2000], Term-modal logics, in R. Dyckhoff, ed., 'Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)', Vol. 1847 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, pp. 220–236.
- GENTZEN G. [1934], 'Untersuchungen über das logische Schließen', *Mathematische Zeitschrift* 39, 176–210, 405–431. Translated as [Gentzen 1969].
- GENTZEN G. [1969], Investigations into logical deduction, in M. Szabo, ed., 'The Collected Papers of Gerhard Gentzen', North Holland, Amsterdam, pp. 68–131. Reprinted from [Gentzen 1934].
- GIRARD J.-Y., LAFONT Y. AND TAYLOR P. [1989], *Proofs and types*, Cambridge University Press.
- GIUNCHIGLIA E., GIUNCHIGLIA F. AND TACCHELLA A. [2000], The SAT-based approach for classical modal logics, in E. Lamma and P. Mello, eds, 'AI*IA 99: Advanced in Artificial Intelligence. 6th Congress of the Italian Association for Artificial Intelligence', Vol. 1792 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Bologna, Italy, pp. 95–106.
- GRISHIN V. [1981], 'Predicate and set-theoretic calculi based on logic without the contraction rule (in Russian)', *Izvestiya Akad. Nauk SSSR* 45(1), 47–68.
- HÄHNLE R. [2001], Tableaux and related methods, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 3, pp. 100–178.
- HINTIKKA K. [1955], 'Form and content in quantification theory', *Acta Philosophica Fennica* 8, 7–55.
- HORROCKS I. AND PATEL-SCHNEIDER P. [1999], 'Optimising description logic subsumption', *Journal of Logic and Computation* 9(3), 267–293.
- HUDELMAIER J. [1993], 'An $O(n \log n)$ -space decision procedure for intuitionistic propositional logic', *Journal of Logic and Computation* 3(1), 63–75.
- KANGER S. [1957], *Provability in Logic*, Vol. 1 of *Studies in Philosophy*, Almqvist and Wicksell, Stockholm.
- KANGER S. [1963], A simplified proof method for elementary logic, in P. Braffort and D. Hirschberg, eds, 'Computer Programming and Formal Systems', North Holland, pp. 87–94. Reprinted as [Kanger 1983].

- KANGER S. [1983], A simplified proof method for elementary logic, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 364–371. Reprinted from [Kanger 1963].
- KETONEN J. AND WEYHRAUCH R. [1984], 'A decidable fragment of predicate calculus', *Theoretical Computer Science* **32**(3), 297–309.
- KLEENE S. [1952], 'Permutability of inferences in Gentzen's calculi LK and LJ', *Memoirs of the American Mathematical Society* **10**, 11–18.
- KLEENE S. [1967], *Mathematical Logic*, John Wiley and Sons.
- KOWALSKI R. AND KUEHNER D. [1971], 'Linear resolution with selection function', *Artificial Intelligence* **2**, 227–260.
- KRIPKE S. [1963], 'Semantical considerations on modal logics', *Acta Philosophica Fennica, Modal and Many-Valued Logics* pp. 83–94.
- KUECHNER D. [1971], A note on the relation between resolution and Maslov's inverse method, in B. Meltzer and D. Michie, eds, 'Machine Intelligence', Vol. 6, American Elsevier, pp. 73–76.
- LETZ R. AND STENZ G. [2001], Model elimination and connection tableau procedures, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 28, pp. 2015–2114.
- LIFSCHITZ V. [1989], 'What is the inverse method?', *Journal of Automated Reasoning* **5**(1), 1–23.
- MAEHARA S. [1954], 'Eine Darstellung der intuitionistischen Logik in der klassischen', *Nagoya Math. J.* **1954**, 45–64.
- MASLOV S. [1964], 'The inverse method of establishing deducibility in the classical predicate calculus', *Soviet Mathematical Doklady* **5**, 1420–1424.
- MASLOV S. [1966], 'An application of the inverse method to the theory of decidable fragments of the classical calculus', *Soviet Mathematical Doklady* **159**(1), 17–20.
- MASLOV S. [1967], 'An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus', *Soviet Mathematical Doklady* **172**(1), 22–25. Reprinted as [Maslov 1983a].
- MASLOV S. [1968], The inverse method of establishing deducibility of logical calculi (in Russian), in 'Collected Works of MIAN', Vol. 98, Nauka, Moscow, pp. 26–87.
- MASLOV S. [1969], 'Relationship between tactics of the inverse method and the resolution method (in Russian)', *Zapiski Nauchnykh Seminarov LOMI* **16**. Reprinted as [Maslov 1983c].
- MASLOV S. [1971a], 'The generalization of the inverse method to predicate calculus with equality (in Russian)', *Zapiski Nauchnykh Seminarov LOMI* **20**, 80–96. English translation in: *Journal of Soviet Mathematics* **1**, no. 1.
- MASLOV S. [1971b], Proof-search strategies for methods of the resolution type, in B. Meltzer and D. Michie, eds, 'Machine Intelligence', Vol. 6, American Elsevier, pp. 77–90.
- MASLOV S. [1983a], An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning (Classical papers on Computational Logic)', Vol. 2, Springer Verlag, pp. 48–54. Reprinted from [Maslov 1967].
- MASLOV S. [1983b], On strategies of resolution and of the inverse method, in M. G., ed., 'Mathematical Logic and Automatic Theorem Proving', Nauka, Moscow, pp. 327–332.
- MASLOV S. [1983c], Relationship between tactics of the inverse method and the resolution method, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning (Classical papers on Computational Logic)', Vol. 2, Springer Verlag, pp. 264–272. Reprinted from [Maslov 1969].
- MASLOV S. AND MINTS G. [1983], The proof-search theory and the inverse method (in Russian), in M. G., ed., 'Mathematical Logic and Automatic Theorem Proving', Nauka, Moscow, pp. 291–314.
- MASLOV S., MINTS G. AND OREVKOV V. [1983], Mechanical proof-search and the theory of logical deduction in the USSR, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning (Classical papers on Computational Logic)', Vol. 1, Springer Verlag, pp. 29–38.
- MINTS G. [1981], Resolution calculi for the non-classical logics (in Russian), in '9th Soviet Symp. on Cybernetics', Vol. 2, VINITI, Moscow, pp. 34–36.

- MINTS G. [1990], Gentzen-type systems and resolution rules. Part I. Propositional logic, in P. Martin-Löf and G. Mints, eds, 'COLOG-88', Vol. 417 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 198–231.
- MINTS G. [1993a], Gentzen-type systems and resolution rule, Part II: Predicate logic, in J. Oikonen and J. Väänänen, eds, 'Proc. ASL Summer Meeting, Logic Colloquium'90', Vol. 2 of *Lecture Notes in Logic*, Springer Verlag, Helsinki, Finland, 15–22 July 1990, pp. 163–190.
- MINTS G. [1993b], 'Resolution calculus for the first order linear logic', *Journal of Logic, Language and Information* 2, 58–93.
- MINTS G. [1994], Resolution strategies for the intuitionistic logic, in 'Constraint Programming', NATO ASI Series F, Springer Verlag, pp. 289–311.
- MINTS G., OREVKOV V. AND TAMMET T. [1996], Transfer of sequent calculus strategies to resolution for S4, in 'Proof Theory of Modal Logic', Studies in Pure and Applied Logic, Kluwer Academic Publishers.
- NIEUWENHUIS R. AND RUBIO A. [2001], Paramodulation-based theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 7, pp. 371–443.
- NONNENGART A. AND WEIDENBACH C. [1998], On generating small clause normal forms, in C. Kirchner and H. Kirchner, eds, 'Automated Deduction — CADE-15. 15th International Conference on Automated Deduction', Vol. 1421 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Lindau, Germany, pp. 397–411.
- NONNENGART A. AND WEIDENBACH C. [2001], Computing small clause normal forms, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 6, pp. 335–367.
- OHLBACH H., NONNENGART A., DE RIJKE M. AND GABBAY D. [2001], Encoding two-valued nonclassical logics in classical logic, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 21, pp. 1403–1486.
- OREVKOV V. [1983], The British museum algorithm may be more efficient than resolution, in M. G., ed., 'Mathematical Logic and Automatic Theorem Proving', Nauka, Moscow, pp. 314–326.
- PAULSON L. [1990], Isabelle: The next 700 theorem provers, in P. Odifreddi, ed., 'Logic and Computer Science', Academic Press, pp. 361–386.
- URL: <http://www.cl.cam.ac.uk/Research/Reports/TR143-lcp-experience.dvi.gz>
- PFENNING F. [2001], Logical frameworks, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 17, pp. 1063–1147.
- PLIUŠKEVIČIUS R. [1965], 'On a variant of the constructive predicate calculus without structural deduction rules', *Soviet Mathematical Doklady* 6, 416–419.
- RAMAKRISHNAN I., SEKAR R. AND VORONKOV A. [2001], Term indexing, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 26, pp. 1853–1964.
- SHANKAR N. [1992], Proof search in the intuitionistic sequent calculus, in D. Kapur, ed., '11th International Conference on Automated Deduction', Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Saratoga Springs, NY, USA, pp. 522–536.
- SMULLYAN R. [1968], *First-Order Logic*, Springer Verlag.
- TAMMET T. [1994], 'Proof strategies in linear logic', *Journal of Automated Reasoning* 12(3), 273–304.
- TAMMET T. [1996], A resolution theorem prover for intuitionistic logic, in M. McRobbie and J. Slaney, eds, 'Automated Deduction — CADE-13', Vol. 1104 of *Lecture Notes in Computer Science*, New Brunswick, NJ, USA, pp. 2–16.
- TAMMET T. [1997], Resolution, inverse method and the sequent calculus, in G. Gottlob, A. Leitsch and D. Mundici, eds, 'Computational Logic and Proof Theory. 5th Kurt Gödel Colloquium, KGC'97', Vol. 1289 of *Lecture Notes in Computer Science*, Vienna, Austria, pp. 65–83.

- TROELSTRA A. S. AND SCHWICHTENBERG H. [1996], *Basic Proof Theory*, Vol. 43 of *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press.
- VORONKOV A. [1985], A proof search method (in Russian), Vol. 107 of *Vychislitelnye Sistemy*, Novosibirsk.
- VORONKOV A. [1990], A proof-search method for the first order logic, in P. Martin-Löf and G. Mintz, eds, 'COLOG'88', Vol. 417 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 327–340.
- VORONKOV A. [1992], Theorem proving in non-standard logics based on the inverse method, in D. Kapur, ed., '11th International Conference on Automated Deduction', Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Saratoga Springs, NY, USA, pp. 648–662.
- VORONKOV A. [1998], 'Proof-search in intuitionistic logic with equality, or back to simultaneous rigid *E*-unification', *Journal of Automated Reasoning* **21**, 205–231.
- VORONKOV A. [1999], KK: a theorem prover for K, in H. Ganzinger, ed., 'Automated Deduction—CADE-16. 16th International Conference on Automated Deduction', *Lecture Notes in Artificial Intelligence*, Trento, Italy, pp. 383–387.
- VORONKOV A. [2000a], How to decide *K* using *X*, in A. Cohn, F. Giunchiglia and B. Selman, eds, 'Principles of Knowledge Representation and Reasoning (KR'2000)', pp. 198–209.
- VORONKOV A. [2000b], How to optimize proof-search in modal logics: a new way of proving redundancy criteria for sequent calculi, in 'LICS'2000'.
- VORONKOV A. [2001], 'How to optimize proof-search in modal logics: new methods of proving redundancy criteria for sequent calculi', *ACM Transactions on Computational Logic*. To appear.
- WAALER A. [2001], Connections in nonclassical logics, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 22, pp. 1487–1578.
- WEIDENBACH C. [2001], Combining superposition, sorts and splitting, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 27, pp. 1965–2013.
- WEIDENBACH C., GAEDE B. AND ROCK G. [1996], SPASS & FLOTTER. Version 0.42, in M. McRobbie and J. Slaney, eds, 'Automated Deduction — CADE-13', Vol. 1104 of *Lecture Notes in Computer Science*, New Brunswick, NJ, USA, pp. 141–145.
- ZAMOV N. [1989], 'Maslov's inverse method and decidable classes', *Annals of Pure and Applied Logic* **42**(2), 165–194.

Index

Symbols

$E(t_1, \dots, t_n)$	186
$E\theta$ — result of application of substitution θ to expression E	186
G -path	234
\square -path	234
$\square\Gamma$	217
$\Gamma \subseteq \Delta$ — Γ is submultiset of Δ	185
\diamond	239
$\Pi\square$	235
$\alpha \cdot \theta$	195
\times	239
\times_*	239
\cup — union of substitutions	186
\diamond -path	234
ϵ	234
\exists -path	256
\forall -path	256
$\dot{\in}$ — multiset membership	185
$\dot{-}$ — multiset difference	185
γ^{mul} — multiset extension of γ	248
\bar{L} — literal complementary to L	203
$\pi \succ \Gamma$	242
\models — equal by definition	186
$\sigma \leq \theta$ — σ is more general than θ	196
$\sigma\theta$ — composition of σ, θ	186
σ_x — substitution obtained by redefinition of σ in x	196
$\sigma _V$ — restriction of substitution σ to the set of variables V	196
$G _{\pi(i)}$	257
γ	240
\triangleright	249
\vee -fork	237
\vee -path	234
\vee_*	239
\wedge -path	234
\wedge_*	239
ξ — goal signed formula	184
$\{\mathbb{P}_{inv}^{G, \gamma}\}$	250
$\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ — notation for substitutions	185

A

admissible substitution	186
Ax_G	224

B

brothers	239
----------	-----

C

C_{inv}^G — free-variable inverse calculus for formulas in NNF	205
C_{inv} — inverse calculus for classical logic	187
C_{inv} for formulas in NNF	205
C_{tab} — tableau calculus for classical logic	190
C_{tab} for formulas in NNF	205
C_{inv}^{ξ} — free-variable inverse calculus for classical logic	198
calculus	249
complete	189
sound	189
clause	222
closed expression	185
\succ -compact path sequent	242
complementary literal	203
complete calculus	189
composition of substitutions	186
conflict	239
contraction	187
contraction-free logics	255

D

D	219
D4	219
derivation	188, 249
diamond-separated	237
disjunctive subformula	227
$dom(\theta)$ — domain of substitution θ	185
domain of substitution	185

E

eigenvariable	187
eigenvariable condition	187
empty substitution	186
expression	185

F

fails (derivation in the set calculus)	249
fair derivation	250
finite rule property	184
formula image	235, 257
$free(E)$ — set of all free variables of E	185
$free(\Gamma)$	197
free immediate signed subformula	192
free immediate subformula	203
free signed subformula	192
free subformula	203

G

G — goal formula	234
G -ordering	239
G_{tab} — tableau calculus for Grishin's logic	255
goal	184
goal formula	234
goal object	249
goal signed formula	184
Grishin's logic	255
ground expression	185
grounding substitution	185

I

I_{inv} — inverse calculus for intuitionistic logic	211
I_{tab} — tableau calculus for intuitionistic logic	210
I_{inv}^{ε} — free-variable inverse calculus for intuitionistic logic	214
immediate signed subformula	192
immediate subformula	203
inference	188
inference rule	249
invertible	189
inference system	249
initial set of objects	249
instance	196
intuitionistic sequent	209
invertible rule	189
\mathbb{P}_{inv}^G	245
$\mathbb{P}_{inv}^{G, \succ}$	246
$\{\mathbb{P}_{inv}^{G, \succ}\}$ — $\mathbb{P}_{inv}^{G, \succ}$ with prefix-subsumption	251

K

K_{inv} — inverse calculus for K	218
K_{inv}^G — free-variable inverse calculus for K	219
K_{tab} — tableau sequent calculus for K	217
$K4$	219

L

limit	250
linear representation of derivation	206
literal	203
complementary	203
literal path	234

M

$\text{mgu}(E_1, E_2)$ — most general unifier	197
$\text{mgu}(\sigma, \tau)$ — most general unifier of substitutions σ, τ	197

modal length	237
model	186
more general substitution	196
most general unifier	197
of substitutions	197
multiset	185
multiset difference	185
multiset extension	248

N

name	221
name calculus for G	222
negation normal form	203
NNF	203

O

objects	249
G -ordering	239
ordering procedure	240

P

parametric signed formulas	188
\mathbb{P}_{tab}^G	235
path	
for first-order logic	256
for propositional modal logic	234
in formula	234
first-order case	256
path calculi	233
path calculus	235
path name	257
path sequent	235
prefix	237
proper	237
prefix-free	237
prefix-free path sequent	258
prefix-subsumes	251
prefix-subsumption rule	251
principal signed formula	188
proper prefix	237

R

$\text{ran}(\theta)$ — range of substitution θ	185
range of substitution	185
rectified formula	195
rectified sequent	195
rectified signed formula	195
redundancy criteria	232
redundancy criterion	206
redundant inferences	232
redundant sequents	232
refutation	188
relevant substitution	195
renaming	197

renaming away	197
renaming rule	197
representation via a free signed subformula	195
represents	195
respect a G -ordering for P_{inv}^G	246
for P_{tab}^G	242
restriction of substitution	196
$Rules_G$	224

S

S4	219
satisfiable	186
satisfiable sequent	189
saturation algorithm	206
sequent	181
of C_{inv}^ξ	196
set calculus	249
side signed formulas	188
sign	181
signed formula	181
simultaneous most general unifier of substitutions	213
sound calculus	189
subformula on path	234
first-order case	256
subformula property	182, 194, 234
submultiset	185
substitution	185
subsumption	248
for path sequents	258
succeeds (derivation in the set calculus)	249

T

T	219
tableau path calculus for Grishin's logic	257
term free for x in E	186
trace	252
true formula	186
true sequent	186

U

unifiable	196
unifier	196
Universal Recipe of Automated Deduction	184

V

$var(E)$ — set of all variables of E	185
$var(G, \pi)$	257
variable range of substitution	185
variable sequence on a path in G	257
variable-disjoint	197

variables of sequent in C_{inv}^ξ	197
variant of sequent in C_{inv}^ξ	197
$varn(\theta)$ — variable range of θ	185

W

weakly unifiable	197
------------------------	-----

Normal Form Transformations

Matthias Baaz

Uwe Egly

Alexander Leitsch

SECOND READERS: Jean Goubault-Larrecq and David Plaisted.

Contents

1	Introduction	275
2	Notation and Definitions	278
3	On the Concept of Normal Form	287
4	Equivalence-Preserving Normal Forms	289
5	Skolem Normal Form	295
6	Conjunctive Normal Form	306
6.1	Nonstructural versus Structural Transformations to CNFs	312
6.2	Comparing Different Structural Transformations	317
6.3	Experimental Results	322
7	Normal Forms in Nonclassical Logics	323
7.1	Intuitionistic Logic	323
7.2	Some Other Nonclassical Logics	327
8	Conclusion	328
	Bibliography	328
	Index	332

1. Introduction

Normal forms traditionally play an important role in mathematics; either they provide information not directly derivable from arbitrary forms (e.g., the rank of matrices) or they allow for more efficient computation (e.g., the vector representation of polynomials). In automated deduction, transformation of first-order formulae to normal form serves the purpose to reduce the inference problem to simple calculi like resolution and paramodulation. A main advantage of such calculi working on normal forms is given by their appropriateness for proof search and heuristics. Indeed, since the beginning of first-order theorem proving in the early sixties, the most efficient theorem provers were those based on clausal normal form. In clausal forms, the whole logical structure of the original formulae is stored in atom formulae and terms, quantifiers and the logical operators are removed.

1.1. EXAMPLE. Let A be the formula

$$[(\forall x)(\exists y)P(x, y) \wedge (\exists u)(\forall v)(P(u, v) \rightarrow Q(u, v))] \rightarrow (\exists w)(\exists z)Q(w, z).$$

Our aim is to give an informal proof of A , i.e., to show the validity of A . We choose proof by contradiction and assume that the antecedent

$$B : (\forall x)(\exists y)P(x, y) \wedge (\exists u)(\forall v)(P(u, v) \rightarrow Q(u, v))$$

holds, but the consequent

$$C : (\exists w)(\exists z)Q(w, z)$$

does not.

As B is true, both formulae $(\forall x)(\exists y)P(x, y)$ and $(\exists u)(\forall v)(P(u, v) \rightarrow Q(u, v))$ are true as well. In particular, the first one (informally) expresses that for every x there is a y such that $P(x, y)$ holds; let us select one particular y for every x and denote it by $f(x)$. Then we may replace $(\forall x)(\exists y)P(x, y)$ by $(\forall x)P(x, f(x))$.

The second formula expresses the existence of an element a for which $(\forall v)(P(a, v) \rightarrow Q(a, v))$ holds. By assumption, C is false, i.e., $\neg C$ is true and so $(\forall w)(\forall z)\neg Q(w, z)$ is true. To sum up, we have reduced the original problem of proving A to the new problem of refuting the set of formulae

$$F : \{(\forall x)P(x, f(x)), (\forall v)(P(a, v) \rightarrow Q(a, v)), (\forall w)(\forall z)\neg Q(w, z)\}.$$

In contrast to the original formula A , no existential quantifiers occur in F . As all variables in F are bound by universal quantifiers, we may delete all occurrences of quantifiers without risk of ambiguity. Thus, F may be replaced by

$$G : \{P(x, f(x)), P(a, v) \rightarrow Q(a, v), \neg Q(w, z)\}.$$

After these reductions, the refutation of G is quite easy: As $P(x, f(x))$ is true for all x , so is the instance $P(a, f(a))$. By substituting $f(a)$ for v in the second formula and applying modus ponens, we obtain $Q(a, f(a))$. This finally contradicts $\neg Q(w, z)$, i.e., the assertion that $Q(w, z)$ is false for all w, z . The contradiction in G finally proves the validity of A .

Example 1.1 demonstrates typical steps in the transformation of formulae to clausal forms and applications of the resolution rule. Instead of G , we would obtain the set of clauses

$$\mathcal{C} : \{\{P(x, f(x))\}, \{\neg P(a, v), Q(a, v)\}, \{\neg Q(w, z)\}\}.$$

The last steps corresponding to the refutation of G can be described by two resolutions using clauses from \mathcal{C} .

In dealing with complex mathematical problems, we frequently represent long formulae or expressions by newly introduced names; a typical sentence corresponding to such a representation is

let A be the formula

This technique (extension by definition) also plays a major role in formula normalization.

1.2. EXAMPLE. Let F be the formula

$$\begin{aligned} &[(\forall x)(\exists y)(P(x, y) \rightarrow (\exists z)Q(y, z)) \wedge \\ &(\forall x)(\exists y)(P(x, y) \rightarrow (\exists z)Q(y, z)) \rightarrow (\exists u)((\forall v)R(u, v) \rightarrow (\forall w)R(u, w))] \rightarrow \\ &(\exists u)((\forall v)R(u, v) \rightarrow (\forall w)R(u, w)). \end{aligned}$$

Like in Example 1.1, it is our aim to prove F . As the formula is quite long, a natural way of preprocessing consists in breaking it apart and naming the parts appropriately. So let us write

$$\begin{aligned} A &\text{ for } (\forall x)(\exists y)(P(x, y) \rightarrow (\exists z)Q(y, z)) \\ B &\text{ for } (\forall x)(\exists y)(P(x, y) \rightarrow (\exists z)Q(y, z)) \rightarrow (\exists u)((\forall v)R(u, v) \rightarrow (\forall w)R(u, w)) \text{ and} \\ C &\text{ for } (\exists u)((\forall v)R(u, v) \rightarrow (\forall w)R(u, w)). \end{aligned}$$

Then F can be represented by $G \rightarrow C$, where G itself can be written as $A \wedge B$; thus F can be written as

$$F_1 : A \wedge B \rightarrow C.$$

Moreover, B itself can be represented as $A \rightarrow C$; inserting the new formula for B into F_1 , we obtain

$$F_2 : [A \wedge (A \rightarrow C)] \rightarrow C.$$

Now we see that it is not necessary to go further "down" in the structure of the formula F ; in fact, F_2 is valid, no matter how the formulae A and C are actually chosen. F_2 can be proven in the same manner as A in Example 1.1; here we obtain the problem of refuting the set

$$\mathcal{D} : \{A, A \rightarrow C, \neg C\},$$

which is even simpler as it is purely propositional.

Example 1.2 illustrates the principle of *structural* or *definitional* transformation; here, the informal naming of formulae is replaced by logical equivalences which then are transformed into clause form. So the introduction of G and C leads to the equivalence $F \leftrightarrow (G \rightarrow C)$; the other equivalences extracted from Example 1.2 are $G \leftrightarrow (A \wedge B)$ and $B \leftrightarrow (A \rightarrow C)$. The clause set $\mathcal{C} : \{\{A\}, \{\neg A, C\}, \{\neg C\}\}$ corresponding to \mathcal{D} can be derived from the clausal forms of the equivalences via resolution.

The examples above indicate that there are several ways of transforming formulae to normal form. As the general aim is to establish efficient methods of proof search, we have to ask which form of normalization guarantees maximal efficiency. In this chapter, we will show that inadequate transformations may lead to problems which have only very long proofs, whereas the right transformations give problems with short proofs. Example 1.2 shows that the relations among subformulae of the formula to be proven may be of central importance in finding a short proof. In proof-theoretic terms, the subformulae may serve as cut formulae in a proof; if the formula is of a rather complex syntax type, the use of such cuts may correspond to an application of substantial and nontrivial lemmata. It is the main purpose of this chapter to compare different methods of normalization with respect to their preservation of *logical structure* and *proof complexity*. We will show that the various methods do not differ only in the length of the computed normal form but differ (much more substantially) in the length of shortest proofs. There are cases where the loss of structure in "nonstructural" normal forms cannot be compensated by clever search strategies and heuristics afterwards; indeed, the lengths of shortest proofs of nonstructural normal forms may be of astronomic dimensions compared with the structural ones. A worst-case analysis shows that the difference is given by the (nonelementary) complexity of cut elimination. This emphasizes the fact that there is no such thing as "the" normal form (e.g., the clausal form) of a formula, but always several ones. Moreover, we have to give a meaningful definition of normal forms at all. First of all, it turns out, as illustrated by the elimination of existential quantifiers in Example 1.1, that preservation of logical equivalence cannot be maintained in general, i.e., it is too restrictive to postulate the logical equivalence of a formula to its normal forms; instead, the natural semantic property is satisfiability-equivalence (or validity-equivalence). Even in cases where logically equivalent normal forms exist, they turn out to be less appropriate to proof length and proof search than the semantically weaker ones. What becomes more important is the *transformation itself* defining the relation among the normalized and the original formula. Consider for instance clausal normal form and satisfiability-equivalence as semantic criterion, i.e., given a formula A , we have to construct a set of clauses \mathcal{C} which is satisfiable iff A is satisfiable. If A is unsatisfiable then clearly the formula $Q(a) \wedge \neg Q(a)$, having the clausal form $\{\{Q(a)\}, \{\neg Q(a)\}\}$, is unsatisfiable too and thus is satisfiability-equivalent to A —independent of the form of A and of the choice of the predicate symbol Q . Obviously, it does not make sense to define $\{\{Q(a)\}, \{\neg Q(a)\}\}$ as clausal normal form of any unsatisfiable formula. The least we have to ask is the computability of the transformation, or better, the computability in polynomial time; this excludes the pathological case where normal

form computations are as costly as theorem proving itself. Moreover, we may ask for a (computationally simple) method to construct a proof (refutation) of the original formula from a proof (refutation) of the normalized one; clearly this is impossible in the example given above.

In Section 5, we discuss Skolem normal form and give a proof theoretic comparison of different skolemization techniques. We prove that prenexing of formulae prior to elimination of quantifiers may be of destructive influence to the existing proofs of the resulting normal forms (nonelementary increase of proof complexity). This property is largely independent of the chosen inference principle and is expressed in terms of Herbrand complexity. In Section 6, we investigate transformations to clausal forms in a similar way. In particular, we compare structural with nonstructural transformations and define different versions of structural normalizations. According to their different power of simulating (analytic) cut, the methods may differ nonelementarily with respect to Herbrand complexity. Moreover, we present experiments which show that structural normalization is superior to the nonstructural one—even in practice. In Section 7, we demonstrate that the principles and techniques of normalization are not limited to classical predicate logic. In particular, we present the Maslov-Mints transformation for intuitionistic predicate logic and give a complexity-theoretic comparison of different normal form transformations for this logic.

2. Notation and Definitions

The set of variables V is defined as the disjoint union of the set of free variables V_f and the set of bound variables V_b . $V(E)$ denotes the set of variables in E ($V_f(E)$ and $V_b(E)$ are defined analogously). We use the letters x, y, z, u, v for bound variables, α and β for free variables. The set of function symbols is denoted by FS, the set of constant symbols by CS and the set of predicate symbols by PS. Constant symbols are represented by a, b, c, d and function symbols by f, g, h . By PL, we denote the set of all formulae in (first-order) predicate logic.

2.1. DEFINITION (*term*). A *term* is defined inductively as follows:

- (i) Each free variable and each constant is a term.
- (ii) If t_1, \dots, t_n are terms and f is an n -place function symbol, then $f(t_1, \dots, t_n)$ is also a term.

The set of all terms is called T .

The set T_s , defined in the same way—with the exception that in (i) $V_f \subseteq T$ is replaced by $V \subseteq T$, is called the set of *semi-terms*. Semi-terms occur naturally by quantification over formulae, see, e.g., Definition 2.10. For details, we refer to [Takeuti 1975].

2.2. DEFINITION (*substitution*). A (variable-) substitution is a mapping $\vartheta : V \rightarrow T_s$ such that $\vartheta(v) \neq v$ for only finitely many $v \in V_f$. We write

$$\vartheta = \{x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n\}$$

for $\vartheta(x_i) = t_i$ and $t_i \neq x_i$, $\vartheta(v) = v$ for $v \notin \{x_1, \dots, x_n\}$. For the composition $\lambda \circ \sigma$ of two substitutions λ and σ , we write $\sigma\lambda$ (postfix notation).

2.3. DEFINITION (*formula*). We define the set of first-order *formulae*: let P be an n -place predicate symbol and t_1, \dots, t_n terms, then $P(t_1, \dots, t_n) \in \text{PL}$ and $P(t_1, \dots, t_n)$ is called an *atom formula*. If $A, B \in \text{PL}$ then $\neg A, (A \wedge B), (A \vee B), (A \rightarrow B), (A \leftrightarrow B) \in \text{PL}$. If $A \in \text{PL}$, $\alpha \in V_f$ and $x \in V_b$ then $(\forall x)A\{\alpha \leftarrow x\}$ and $(\exists x)A\{\alpha \leftarrow x\}$ are in PL (we allow dummy quantification, i.e., the case where α does not occur in A).

2.4. EXAMPLE. Let f be a two-place function symbol. Then $f(\alpha, f(\alpha, a))$ is a term, $f(x, f(\alpha, a))$ is a semi-term. Note that the atom formula $P(f(\alpha, a), \alpha)$ contains only terms, while the formula $(\exists x)P(f(\alpha, a), x)$ also contains semi-terms.

Substitutions are extended to terms, semi-terms and formulae in the obvious way. Substitutions are applied to formulae in postfix notation too, i.e., instead of $\sigma(A)$, we write $A\sigma$.

We sometimes need the notion of an *occurrence of a formula*. Occurrences can be formally defined as sequences of natural numbers indicating the position of a subformula within the formula tree. Occasionally, we have to replace (occurrences of) formulae within formulae by other ones (this operation is not a substitution). Let B be the subformula occurring at position λ in A . Then $A[C]_\lambda$ denotes A after the replacement of B by C at position λ in A . If it is clear which position we really mean, we also write $A[C]$ instead of $A[C]_\lambda$.

2.5. DEFINITION. Let A be a formula in PL containing the free variables $\alpha_1, \dots, \alpha_n$ and let $x_1, \dots, x_n \in V_b$. Then the formula

$$(\forall x_1) \dots (\forall x_n) A\{\alpha_1 \leftarrow x_1, \dots, \alpha_n \leftarrow x_n\}$$

is called a *universal closure* of A and it is denoted by $\forall A$.

2.6. DEFINITION. Let $A, B \in \text{PL}$. We define

- $A \sim B$, if A and B are logically equivalent,
- $A \sim_{\text{sat}} B$, if A and B are satisfiability equivalent, i.e., A is satisfiable iff B is satisfiable.
- $A \sim_{\text{val}} B$, if A and B are validity equivalent, i.e., A is valid iff B is valid.

Clearly, $A \sim B$ implies $A \sim_{\text{sat}} B$ and $A \sim_{\text{val}} B$.

2.7. DEFINITION (*polarity*). Let λ be an occurrence of a formula A in B . If $A = B$ then λ is a positive occurrence in B . If $B = (C \wedge D)$, $B = (C \vee D)$, $B = (\forall x)C$ or $B = (\exists x)C$ and λ is a positive (negative) occurrence of A in C (or in D , respectively) then the corresponding occurrence λ' of A in B is positive (negative). If $B = (C \rightarrow D)$ and λ is a positive (negative) occurrence of A in D then the corresponding occurrence λ' in B is positive (negative); if, on the other hand, λ is a positive (negative) occurrence of A in C then the corresponding occurrence λ' of A in B is negative (positive). If $B = \neg C$ and λ is a positive (negative) occurrence of A in C then the corresponding occurrence λ' of A in B is negative (positive). If there exists a positive (negative) occurrence of a formula A in B , we say that A is of positive (negative) polarity in B .

If λ is a positive (negative) occurrence of a formula $(Qx)A$ in B (for $Q \in \{\forall, \exists\}$) then we say that (Qx) occurs positively (negatively) in B .

2.8. DEFINITION (*strong and weak quantifiers*). If $(\forall x)$ occurs positively (negatively) in B then $(\forall x)$ is called a strong (weak) quantifier. If $(\exists x)$ occurs positively (negatively) in B then $(\exists x)$ is called a weak (strong) quantifier.

Note that (Qx) may occur several times in a formula B ; thus, it may be strong and weak at the same time. If confusion might arise, we refer to the specific position of (Qx) in B . Particularly, we may replace every formula A by a logically equivalent "variant" A' such that every (Qx) (for $Q \in \{\forall, \exists\}$ and $x \in V_b$) occurs at most once in A' . We call such formulae *rectified* (see Definition 4.8) in Section 4. In this case, the term " (Qx) is a strong (weak) quantifier" has a unique meaning.

2.9. DEFINITION (*sequent*). A sequent is an expression of the form $\Gamma \vdash \Delta$ where Γ and Δ are finite multisets of PL-formulae (i.e., two sequents $\Gamma_1 \vdash \Delta_1$ and $\Gamma_2 \vdash \Delta_2$ are considered equal if the multisets represented by Γ_1 and by Γ_2 are equal and those represented by Δ_1 and by Δ_2 are equal too).

If $S = A_1, \dots, A_n \vdash B_1, \dots, B_m$ then we say that the A_i occur negatively and the B_j occur positively in S . A quantifier occurring positively (negatively) in A_i occurs negatively (positively) in S . Likewise, a quantifier occurring positively (negatively) in B_j occurs positively (negatively) in S .

S is called closed if all A_i, B_j are closed.

2.10. DEFINITION (*the calculus LK*). We define the calculus LK essentially like [Takeuti 1975]. In particular we do not treat the connective \leftrightarrow but confine ourselves to the connectives $\{\neg, \wedge, \vee, \rightarrow\}$. The difference consists in working with multisets instead of sequences and in the role of contractions in the rules. The initial sequents are of the form $A \vdash A$ for first-order formulae A not containing \leftrightarrow . In defining the rules of LK, we always mark the auxiliary formulae of an inference and the principal formula (by another marking symbol). Thus, in our definition, \wedge -introduction to the right takes the form

$$\frac{\Gamma_1 \vdash A^+, \Delta_1 \quad \Gamma_2 \vdash \Delta_2, B^+}{\Gamma_1, \Gamma_2 \vdash \Delta_1, (A \wedge B)^*, \Delta_2}$$

We usually avoid markings by putting the auxiliary formulae at the leftmost position in the antecedent and in the rightmost position in the consequent. The principal formula mostly is identifiable by the context. Thus, the rule above will be written as

$$\frac{\Gamma_1 \vdash \Delta_1, A \quad \Gamma_2 \vdash \Delta_2, B}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, A \wedge B}$$

By the definition of sequents over multisets, we do not need the exchange rules. In our notation Γ, Δ, Π and Λ serve as metavariables for sequences of formulae; \vdash is the separation symbol.

1. The logical rules:

$$\frac{\Gamma \vdash \Delta, A \quad \Pi \vdash \Lambda, B}{\Gamma, \Pi \vdash \Delta, \Lambda, A \wedge B} \wedge : r$$

$$\frac{A, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta} \wedge : l1$$

$$\frac{A, \Gamma \vdash \Delta}{B \wedge A, \Gamma \vdash \Delta} \wedge : l2$$

$$\frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B} \vee : r1$$

$$\frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, B \vee A} \vee : r2$$

$$\frac{A, \Gamma \vdash \Delta \quad B, \Pi \vdash \Lambda}{A \vee B, \Gamma, \Pi \vdash \Delta, \Lambda} \vee : l$$

$$\frac{\Gamma \vdash \Delta, A \quad B, \Pi \vdash \Lambda}{A \rightarrow B, \Gamma, \Pi \vdash \Delta, \Lambda} \rightarrow : l$$

$$\frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \rightarrow B} \rightarrow : r$$

$$\frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A} \neg : r$$

$$\frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta} \neg : l$$

$$\frac{\Gamma \vdash \Delta, A\{x \leftarrow \alpha\}}{\Gamma \vdash \Delta, (\forall x)A} \forall : r$$

$$\frac{A\{x \leftarrow t\}, \Gamma \vdash \Delta}{(\forall x)A, \Gamma \vdash \Delta} \forall : l$$

$\forall : r$ must fulfil the eigenvariable condition, i.e., the free variable α does not occur in $\Gamma \vdash \Delta, (\forall x)A$. In $\forall : l$, t may be an arbitrary term (note that terms do not contain bound variables!). $\forall : r$ is called a strong, $\forall : l$ a weak quantifier introduction. The conditions for $\exists : r$ are the same as for $\forall : l$ and similarly for $\exists : l$ and $\forall : r$.

$$\frac{\Gamma \vdash \Delta, A\{x \leftarrow t\}}{\Gamma \vdash \Delta, (\exists x)A} \exists : r$$

$$\frac{A\{x \leftarrow \alpha\}, \Gamma \vdash \Delta}{(\exists x)A, \Gamma \vdash \Delta} \exists : l$$

2. The structural rules:
weakenings:

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A} w : r \qquad \frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta} w : l$$

contractions:

$$\frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A} c : r \qquad \frac{A, A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta} c : l$$

$$\frac{\Gamma \vdash \Delta, A \quad A, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda} cut$$

The formula A occurring in the cut rule is called the *cut formula* of the inference.

2.11. DEFINITION. An LK-*proof* is defined as a (directed) tree, where the nodes are occurrences of sequents and the edges are defined according to the rules applied to the sequents. The leaves of a proof ω are occurrences of *initial sequents* $A \vdash A$, the sequent occurring at the root is called *end sequent*. If ψ is an LK-proof of the sequent $\vdash A$ for $A \in \text{PL}$, we also call ψ a proof of A .

An important subset of LK-proofs is the set of *cut-free proofs*, i.e., the proofs without occurrence of the cut rule. Cuts can always be eliminated (algorithmically) from LK-proofs and thus cut-free proofs define a complete subcalculus of LK. LK with *analytic cut* lies between cut-free and full LK; in this subcalculus, cuts are allowed, but the cut formulae must occur as subformulae in the end sequent. To this aim, we need the following weak subformula property expressed by the binary relation \prec : If $A = B \circ C$ for $\circ \in \{\wedge, \vee, \neg, \rightarrow, \}$ then $B \prec A$ and $C \prec A$. If $A = \neg B$ then $B \prec A$. If $A = (Qx)B$ for $Q \in \{\forall, \exists\}$ then, for all terms t , $B\{x \leftarrow t\} \prec A$. Let \preceq be the reflexive and transitive closure of \prec . Then B is a (weak) subformula of A if $B \preceq A$. Due to the definition of subformulae of quantified formulae, there may be infinitely many B s with $B \preceq A$.

2.12. EXAMPLE. Let A be the formula $(\forall x)(\exists y)(P(x) \rightarrow Q(y))$. Then

$$B : (\exists y)(P(f(a)) \rightarrow Q(y)) \text{ and } C : P(f(a)) \rightarrow Q(b)$$

are weak subformulae of A . Indeed, $B \prec A$, $C \prec B$, and so $C \preceq A$.

Let ψ be an LK-proof of $Q(f(f(a))) \rightarrow A$ with cut formulae B and C . Then ψ is a proof with analytic cuts.

Paths in a proof are defined in the traditional way. A *branch* is a path starting in the end sequent. The maximal length of a branch in an LK-proof ψ is called the *height* of ψ and is denoted by $h(\psi)$.

We use the terms "predecessor" and "successor" in the intuitive sense (i.e., contrary to the direction of edges in the tree): If there exists a path from S_1 to S_2 then S_2 is called a *predecessor* of S_1 ; particularly, every sequent (occurrence) is predecessor of the end sequent. The successor relation is defined in an analogous way. The predecessor relation and the successor relation are extended to occurrences of formulae in sequents.

2.13. EXAMPLE. Let ψ be the following LK-proof, where the μ_i ($0 \leq i \leq 4$) indicate the occurrences of sequents

$$\frac{\frac{\mu_2 : P(a) \vdash P(a)}{\mu_1 : (\forall x)P(x) \vdash P(a)} \forall : l \quad \frac{\mu_4 : P(b) \vdash P(b)}{\mu_3 : (\forall x)P(x) \vdash P(b)} \forall : l}{\mu_0 : (\forall x)P(x), (\forall x)P(x) \vdash P(a) \wedge P(b)} \wedge : r$$

Then all μ_i are predecessors of μ_0 , μ_2 is predecessor of μ_1 , but not of μ_3 . The occurrence μ_4 is predecessor of μ_3 and of μ_0 , but not of μ_1, μ_2 .

Our complexity analysis aims at the distinction between transformations of elementary and those of nonelementary complexity. A specific role play the bound functions e and s :

2.14. DEFINITION. Let $e : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ be defined as:

$$\begin{aligned} e(0, n) &= n \text{ for } n \in \mathbb{N}, \\ e(m+1, n) &= 2^{e(m, n)} \text{ for } m, n \in \mathbb{N}. \end{aligned}$$

A function $g : \mathbb{N} \rightarrow \mathbb{N}$ is called *elementary* if there exists a Turing machine M computing g and a number $k \in \mathbb{N}$ (in a unary coding of the tape) with

$$\text{time}_M(n) \leq e(k, n) \text{ for all } n \in \mathbb{N},$$

where $\text{time}_M(n)$ is the computing time of M on input n .

2.15. REMARK. The set of elementary functions remains unchanged if we define it via Turing machines using an n -ary coding of the tape (for arbitrary $n \in \mathbb{N}$). Thus, the set is largely independent of syntactic representations of the input. For a detailed description of Turing machines and computability, see [Hopcroft and Ullman 1979].

2.16. DEFINITION. Let e be as in Definition 2.14. Then $s : \mathbb{N} \rightarrow \mathbb{N}$ is defined as $s(n) = e(n, 1)$ for all $n \in \mathbb{N}$.

It is easy to see that s is not an elementary function; indeed, for all numbers $k \in \mathbb{N}$, the inequality $s(n) \leq e(k, n)$ only holds for finitely many n , and $s(n) \leq \text{time}_M(n)$ holds for every Turing machine M computing s . For the last inequality, we assumed unary coding of the input; however, by $2^{s(n)} = s(n+1)$, the situation is the same even for binary coding.

In the remaining part of this section, we give some basic definitions concerning clause logic and resolution.

We define the notions *atom*, *literal*, *expression* and *clause* formally.

2.17. DEFINITION (*atom*). Let P be an n -place predicate symbol and t_1, \dots, t_n semi-terms then $P(t_1, \dots, t_n)$ is called an *atom*.

Note that, as clause logic is quantifier-free, the distinction between free and bound variables becomes inessential. Therefore, we allow arbitrary variables to occur in atoms and literals.

2.18. DEFINITION (*literal*). A *literal* is either an atom or an atom preceded by a negation sign. The *set of arguments* of a literal L is denoted by $\text{args}(L)$.

2.19. DEFINITION (*expression*). An *expression* is either a term, a semi-term, or a literal.

2.20. DEFINITION (*clause*). A *clause* is a finite set of literals. The *empty clause* is denoted by \square .

Usually, we write clauses as disjunctions, e.g., instead of $\{Q(y), P(a, y)\}$ we write $Q(y) \vee P(a, y)$.

2.21. DEFINITION. If a literal L is unsigned, i.e., if it is identical with an atom A , then the *dual* of L , denoted by L^d , equals $\neg A$. Otherwise, if L is of the form $\neg A$ then $L^d = A$. For a set of literals $C = \{L_1, \dots, L_n\}$, we define $C^d = \{L_1^d, \dots, L_n^d\}$. The dual of a quantifier Q , denoted by Q^d , is \forall if Q is \exists , and \exists otherwise.

Additionally, we introduce the following notation:

2.22. DEFINITION. C_+ is the set of positive (unsigned) literals of a clause C ; analogously, C_- denotes the set of negative literals (negated atoms) in C .

2.23. DEFINITION. C is a *Horn clause* iff it contains at most one positive literal, i.e., $|C_+| \leq 1$.

If S is some set of expressions, clauses or clause sets then $\text{CS}(S)$, $\text{FS}(S)$, and $\text{PS}(S)$, refers to the set of constant symbols, function symbols and predicate symbols, respectively, that occur in S .

The term depth of an expression or a clause is defined as follows:

2.24. DEFINITION. The *term depth* of a semi-term t , denoted by $\tau(t)$, is defined by:

- (i) If t is a variable or a constant, then $\tau(t) = 0$.
- (ii) If $t = f(t_1, \dots, t_n)$, where f is an n -place function symbol, then $\tau(t) = 1 + \max\{\tau(t_i) \mid 1 \leq i \leq n\}$.

The term depth of a literal L is defined as $\tau(L) = \max\{\tau(t) | t \in \text{args}(L)\}$. The term depth of a clause C is defined as $\tau(C) = \max\{\tau(L) | L \in C\}$. For a set S of clauses, we define $\tau(S) = \max\{\tau(C) | C \in S\}$.

We define E_1 and E_2 to be *variable-disjoint* iff $V(E_1) \cap V(E_2) = \emptyset$.

2.25. DEFINITION. An expression or a clause is called *ground*, if no variables occur in it. We call it *constant-free*, if no constants occur in it, and *function-free*, if it does not contain function symbols.

2.26. DEFINITION. A *ground substitution* is a substitution σ such that there are only ground terms in the range of σ .

2.27. DEFINITION. An expression E_1 is an *instance* of another expression E_2 iff there exists a substitution σ such that $E_1 = E_2\sigma$. Likewise, a clause C_1 is an instance of clause C_2 iff $C_1 = C_2\sigma$ for some substitution σ .

We may compare expressions, substitutions and clauses using the following ordering relation.

2.28. DEFINITION. Let E_1 and E_2 be expressions. Then $E_1 \leq_s E_2$ —read: E_1 is *more general than* E_2 —iff there exists a substitution σ such that $E_1\sigma = E_2$. For substitutions ρ and θ , we define analogously: $\rho \leq_s \theta$ iff there exists a substitution σ such that $\rho\sigma = \theta$. Similarly, if C and D are clauses, $C \leq_s D$ iff there exists a substitution σ such that $C\sigma \subseteq D$. In this case, we also say, in accordance with the usual resolution terminology, that C *subsumes* D .

2.29. DEFINITION. A set of expressions M is *unifiable* by a substitution σ iff $E_i\sigma = E_j\sigma$ for all $E_i, E_j \in M$. σ is called *most general unifier (m.g.u.)* of M iff for every other unifier ρ of M : $\sigma \leq_s \rho$. We shall also say that E_1 is *unifiable with* E_2 iff $\{E_1, E_2\}$ is unifiable.

Remember that any two different most general unifiers of a set of expressions only differ in the names of the variables.

For the *resolvent*, we retain the original definition of Robinson [Robinson 1965] which combines factorization and (binary) resolution.

2.30. DEFINITION (*resolvent*). Let C, D be clauses and C' and D' be variable-disjoint variants of C and D . Moreover let M and N be nonempty subsets of C' and of D' respectively, such that $N^d \cup M$ is unifiable by the m.g.u. θ . Then $E = (C' - M)\theta \cup (D' - N)\theta$ is a *Robinson-resolvent* of C and D .

If M and N are singleton sets then E is called *binary resolvent* of C and D .

2.31. DEFINITION (*resolution derivation*). A *resolution derivation* of a clause C from a set of clauses \mathcal{C} is a sequence C_1, \dots, C_n such that $C_n = C$ and, for all $m \leq n$, either $C_m \in \mathcal{C}$ or there exist $i, j < m$ such that C_m is a resolvent of C_i and C_j . A resolution derivation of \square from \mathcal{C} is called a *refutation* of \mathcal{C} .

In the definition above, resolution derivations are formulated in a dag-type manner, i.e., in form of a directed acyclic tree. Another standard representation is by tree structures (like for LK-proofs).

2.32. DEFINITION (tree resolution). A *resolution tree* Φ is a binary tree defined by a set of nodes labelled by clauses; if (N, N_1) and (N, N_2) are edges in Φ and N, N_1, N_2 are labelled by the clauses C, C_1, C_2 then C is a resolvent of C_1 and C_2 . If the root node of Φ is labelled by C and the leaves are labelled by $\{C_1, \dots, C_n\}$ then Φ is called a *tree resolution derivation* of C from $\{C_1, \dots, C_n\}$. A tree resolution derivation of \square is called *tree resolution refutation* (or tree refutation for short).

2.33. EXAMPLE. Let \mathcal{C} be the set of clauses $\{P(a), \neg P(x) \vee P(f(x)), \neg P(f(f(a)))\}$. Then the sequence ϕ

$$P(a), \neg P(x) \vee P(f(x)), \neg P(y) \vee P(f(f(y))), P(f(f(a))), \neg P(f(f(a))), \square$$

is a resolution refutation of \mathcal{C} . Note that the third clause is obtained by a self-resolution of the second one (rename x to y and resolve the literals $\neg P(x)$ and $P(f(y))$). The corresponding tree refutation Φ (written in LK-like format) is:

$$\frac{\frac{\neg P(x) \vee P(f(x)) \quad \neg P(x) \vee P(f(x))}{\neg P(y) \vee P(f(f(y)))} \quad P(a)}{\frac{P(f(f(a))) \quad \neg P(f(f(a)))}{\square}}$$

Note that ϕ consists of 6 clauses, while Φ has 7 nodes. In the worst case, shortest tree refutations may be exponential in shortest (dag-) resolution refutations. Some derivations can be turned into ground derivations by replacing the clauses by appropriate ground instances. Such a transformation is called *ground projection* [Baaz and Leitsch 1992]. A ground projection (the only possible one) of Φ in the example above is:

$$\frac{\frac{\neg P(a) \vee P(f(a)) \quad \neg P(f(a)) \vee P(f(f(a)))}{\neg P(a) \vee P(f(f(a)))} \quad P(a)}{\frac{P(f(f(a))) \quad \neg P(f(f(a)))}{\square}}$$

The resolution derivation ϕ does not have ground projections at all. I.e., there exists no replacement of the sequence $\phi : C_1, \dots, C_5, \square$ by a sequence $\phi' : C'_1, \dots, C'_5, \square$ such that the C'_i are ground instances of the C_i and ϕ' is a resolution refutation of the set of ground instances of \mathcal{C} . On the other hand, tree resolution refutations always have ground projections. For a closer analysis of ground projections of resolution derivations, see [Baaz and Leitsch 1992].

3. On the Concept of Normal Form

Simplification of formulae or terms plays a central role in mathematics and computer science. The aim of simplifications usually are *normal forms*, i.e., formulae or terms belonging to a specific syntactically simple subclass. Perhaps the simplest example is the relation among an arithmetical term and its value, e.g., the term $1 + 2$ evaluates to 3. If we represent this problem in the logical theory of arithmetic, we obtain $s(s(s(0)))$ as the normal form of $s(0) + s(s(0))$. The rules of transforming arithmetical ground terms to their normal form (corresponding to the result of the represented computation) are simply the rules of elementary arithmetic. In mathematical logic, normal forms cannot be viewed as "results" in such a straightforward way (consider, e.g., the conjunctive normal form $(\neg A \vee B) \wedge (A \vee \neg B)$ of $A \leftrightarrow B$). Intuitively, it is deduction which plays the role of computation in mathematical logic. Nevertheless, the process of normalization, i.e., the computation of normal forms, has several very general features which are common to many areas ranging from interpretation of LISP-programs to arithmetical normalizations and normal form computations in automated deduction. The central concept of any theory of normalization is *reduction*. Formally, a reduction relation is an irreflexive (binary) relation \mapsto_N on some domain; as our focus of interest is predicate logic, we choose the set of first-order formulae PL.

3.1. DEFINITION. A reduction relation is an irreflexive relation on the set of first-order formulae.

If \mapsto_N is a reduction relation, we write $A \mapsto_N B$ for $(A, B) \in \mapsto_N$. In this case, we say that A *reduces* to B .

3.2. DEFINITION. Let \mapsto_N be a reduction relation on PL. A formula A in PL is in \mapsto_N -normal form (or, shorthand, in normal form) if there exists no formula B with $A \mapsto_N B$. We also say that A is *irreducible* under \mapsto_N .

If a formula is not in normal form, we may apply the relation \mapsto_N iteratively until we eventually obtain an irreducible formula A ; this is the essence of a *normal form computation*.

3.3. DEFINITION (normal form). Let \mapsto_N be a reduction relation and \mapsto_N^* be the reflexive and transitive closure of \mapsto_N . Let $A \mapsto_N^* B$ and B is irreducible under \mapsto_N ; then we call B a *normal form* of A .

Note that the definition above is quite general; it neither requires the existence nor the uniqueness of normal forms.

So far, the definitions above only address the syntactical part of normalization, but they tell nothing about correctness. Clearly, we could define $A \mapsto_N P(a)$ for all formulae A different from $P(a)$ (and $P(a) \not\mapsto_N B$ for all formulae B). According to this definition, every first-order formula A has $P(a)$ as its normal form, no matter whether A is valid, satisfiable or unsatisfiable. Obviously, \mapsto_N does not make sense

as it does not obey even the weakest semantical constraints; we have to ask for preservation of logical equivalence or, at least, of satisfiability or validity.

3.4. DEFINITION. Let \mapsto_N be a reduction relation on PL. \mapsto_N is called *strongly correct* if, for all formulae A, B , $A \mapsto_N B$ implies $A \sim B$; it is called *val-correct* if $A \mapsto_N B$ implies $A \sim_{val} B$ and it is called *sat-correct* if $A \mapsto_N B$ implies $A \sim_{sat} B$.

3.5. EXAMPLE. Let A be a formula and B be a subformula of A occurring at position λ . If $B = \neg(C \wedge D)$ then we define

$$A \mapsto_{N_1} A[\neg C \vee \neg D]_\lambda,$$

i.e., \mapsto_{N_1} reduces A by applying a rule of de Morgan at any position which admits such a rule application. Otherwise, \mapsto_{N_1} is undefined. Clearly, de Morgan's rules are (strongly) correct and thus $\neg(C \wedge D) \sim \neg C \vee \neg D$. Moreover, replacing subformulae by logically equivalent ones preserves logical equivalence; thus $A \mapsto_{N_1} B$ implies $A \sim B$ and \mapsto_{N_1} is strongly correct.

The formula $F : P(a) \wedge \neg(P(b) \vee P(c))$ is in \mapsto_{N_1} -normal form as it is irreducible. The formula $G : P(a) \vee \neg(P(b) \wedge P(c))$ is not in \mapsto_{N_1} -normal form, but G reduces to $H : P(a) \vee (\neg P(b) \vee \neg P(c))$. H is irreducible and the normal form of G with respect to \mapsto_{N_1} .

3.6. EXAMPLE. Let F be a closed formula of the form $(\exists v)G$ for a variable v and a formula G . We define

$$F \mapsto_{N_2} G\{v \leftarrow a\}$$

where a is a constant symbol not occurring in F ; otherwise, \mapsto_{N_2} is undefined. In contrast to the relation \mapsto_{N_1} in Example 3.5, \mapsto_{N_2} is not strongly correct: $(\exists x)P(x) \mapsto_{N_2} P(a)$, but $(\exists x)P(x) \not\sim P(a)$. The relation \mapsto_{N_2} is not even val-correct, because $(\exists x)(P(x) \vee \neg P(a)) \mapsto_{N_2} P(b) \vee \neg P(a)$, where the first formula is valid but the second is not. In particular, $A \mapsto_{N_2} B$ does not imply $A \sim_{val} B$.

On the other hand, \mapsto_{N_2} is sat-correct, because $(\exists v)G$ is satisfiable iff $G\{x \leftarrow a\}$ is satisfiable. The relation \mapsto_{N_2} defines a limited form of skolemization, which will be discussed in detail in Section 5.

For computational purposes, it is important that any reduction chain defined by a reduction relation \mapsto_N terminates, i.e., any normal form computation yields a result. Termination is a property which, in general, cannot be achieved for reduction relations modelling arbitrary computations (e.g., β -reduction in the λ -calculus). But for meaningful logical normalization, which typically serves as a preprocessing, termination is absolutely necessary.

3.7. DEFINITION. A reduction relation \mapsto_N is called *terminating* if there exists no infinite sequence of formulae $(A_i)_{i \in \mathbb{N}}$ with $A_i \mapsto_N A_{i+1}$ for all $i \in \mathbb{N}$.

Note that the relations \rightarrow_{N_1} and \rightarrow_{N_2} in the examples above are both terminating. There are strongly correct reduction relations which do not define any normal form.

3.8. EXAMPLE. Let F be a formula and A be a subformula of F occurring at position λ ; then we define

$$F[A]_{\lambda} \rightarrow_N F[\neg A]_{\lambda}.$$

The relation \rightarrow_N is nonterminating and, even worse, every reduction sequence is infinite! In fact, for any formula B

$$B \rightarrow_N \neg B \rightarrow_N \neg\neg B \rightarrow_N \dots$$

Therefore, there exists no irreducible formula and thus no formula has a normal form. Clearly, the reduction makes sense only if we turn it around:

$$F[\neg A]_{\lambda} \rightarrow_M F[A]_{\lambda}.$$

The relation \rightarrow_M is strongly correct and terminating; moreover, the resulting normal form is unique.

In general (e.g., for prenex normal forms), we may have $A \rightarrow_N^* B$ and $A \rightarrow_N^* C$, where B and C are in normal form, but $B \neq C$; in this case, B and C are different normal forms of A . In particular, different normal form computations may yield different results. For computational purposes, it is necessary to make a deterministic choice, i.e., to decide for a specific normal form computation.

3.9. DEFINITION (*normal form operator*). A normal form operator ν based on a reduction relation \rightarrow_N is a computable mapping $\text{PL} \rightarrow \text{PL}$ such that $\nu(A) = B$ implies that B is a \rightarrow_N -normal form of A .

If \rightarrow_N defines unique normal forms then $\nu(A) = B$ iff B is *the* normal form of A . Normal form operators based on correct, val-correct and sat-correct reduction relations correspond to the typical normal form computations in automated deduction. This chapter is mainly devoted to the investigation of such operators (computing skolem forms, conjunctive normal form etc.) and their relation to proof complexity.

4. Equivalence-Preserving Normal Forms

In Section 3, we have defined different semantic types of normal forms. The most natural, although usually not the most useful one, is the type of strongly correct normal form reductions. Perhaps the best known normal forms are the conjunctive normal form (CNF) and the disjunctive normal form (DNF). In propositional logic, it is possible to transform every formula into CNF or DNF under preservation of logical equivalence (note that even here val-correct or sat-correct normal forms are preferable from the point of computational complexity). In predicate logic, the

construction of CNF or DNF requires the elimination of quantifiers, a task which necessarily destroys logical equivalence. However, also in predicate logic, there are several strongly correct normalizations; most of them are rather of metatheoretic than of computational value. We only present two of them, negation normal form (NNF) and prenex normal form (PNF).

4.1. EXAMPLE. Let A be the formula

$$\neg((\forall x)\neg P(x) \vee Q(b)).$$

A contains a disjunction which is "essentially" a conjunction; also the universal quantifier has the meaning of an existential one. This is a consequence of the negation sign which dominates the subformulae $(\forall x)\neg P(x)$ and $(\forall x)\neg P(x) \vee Q(b)$. If we want logical operators to express their traditional meaning, we have to remove negative polarity as far as possible. Using de Morgan's law first, we transform A into

$$B : \neg(\forall x)\neg P(x) \wedge \neg Q(b).$$

In B , the universal quantifier is still in the scope of a negation. By applying the (strongly correct) reduction

$$\neg(\forall v)F \mapsto (\exists v)\neg F,$$

we obtain the formula

$$C : (\exists x)\neg\neg P(x) \wedge \neg Q(b).$$

By the strongly correct reduction $\neg\neg F \mapsto F$, we eventually obtain

$$D : (\exists x)P(x) \wedge \neg Q(b).$$

Now, negation only occurs in front of the atom $Q(b)$; indeed, $\neg Q(b)$ is the only subformula of D under a negation sign.

Example 4.1 illustrates the construction of a negation normal form (NNF), where atoms are the only subformulae which are of negative polarity. In the general transformation to NNF, we have to remove implications (where the antecedent is of negative polarity) and the equivalence \leftrightarrow (where the connected formulae are "bipolar").

4.2. DEFINITION (*negation normal form*). A formula A is in negation normal form if

1. A does not contain the connectives \rightarrow , \leftrightarrow and
2. if $\neg B$ is a subformula of A then B is an atom.

4.3. DEFINITION. We define a reduction relation \mapsto_ν in the following way:

$$\text{nfn-1 } (A \rightarrow B) \mapsto_\nu (\neg A \vee B),$$

$$\text{nfn-2 } (A \leftrightarrow B) \mapsto_\nu ((\neg A \vee B) \wedge (A \vee \neg B)),$$

nnf-3 $\neg(\forall v)A \mapsto_\nu (\exists v)\neg A, \neg(\exists v)A \mapsto_\nu (\forall v)\neg A;$

nnf-4 $\neg\neg A \mapsto_\nu A,$

nnf-5 $\neg(A \wedge B) \mapsto_\nu (\neg A \vee \neg B), \neg(A \vee B) \mapsto_\nu (\neg A \wedge \neg B),$

nnf-6 $\neg(A \rightarrow B) \mapsto_\nu (A \wedge \neg B),$

nnf-7 $\neg(A \leftrightarrow B) \mapsto_\nu ((A \wedge \neg B) \vee (\neg A \wedge B)).$

The reduction by \mapsto_ν can be applied on every subformula, i.e., we extend \mapsto_ν in the following way: Let λ be a position in a formula F and G be the subformula occurring at λ and $G \mapsto_\nu H$. Then we define $F \mapsto_\nu F[H]_\lambda$.

Applications of \mapsto_ν shift negation "downwards" in the formula tree as long as possible; all possible transformations are terminating, no matter which order is chosen.

4.4. PROPOSITION. \mapsto_ν is terminating.

PROOF. We do not give a full proof, but merely consider formulae which do not contain the connectives $\rightarrow, \leftrightarrow$.

It is sufficient to find a measure which is strictly decreased by \mapsto_ν . Let F be a formula and Λ be the set of all positions of negations in F . For every $\lambda \in \Lambda$, we define $q(\lambda)$ as the number of occurrences of logical operators below λ . Then we set $m(F)$ to the sum of all $q(\lambda)$ for $\lambda \in \Lambda$. Clearly, $F \mapsto_\nu G$ implies $m(F) > m(G)$ in all cases nnf-3 to nnf-5. Therefore, the number of possible reductions on a formula F not containing the connectives \rightarrow and \leftrightarrow is bounded by $m(F)$ and \mapsto_ν is terminating. In order to get a full proof, we have to extend the m -ordering to a tuple ordering where the first component contains the number of connectives in $\{\rightarrow, \leftrightarrow\}$. \square

Note that we could have refined the relation \mapsto_ν by splitting it into two reduction relations: the first one \mapsto_1 , consisting of nnf-1 and nnf-2 only, eliminates \rightarrow and \leftrightarrow , the second one \mapsto_2 (defined by nnf-3 to nnf-5) serves the purpose to shift \neg downwards. Then the proof above shows termination of \mapsto_2 on \mapsto_1 -normal forms. The termination of \mapsto_1 is trivial as the number of connectives $\in \{\rightarrow, \leftrightarrow\}$ decreases in every reduction step. It is easy to see that nnf-6 and nnf-7 are not really necessary (but merely convenient); these rules can be simulated by nnf-1 to nnf-5. Indeed, any normal form under nnf-1 to nnf-5 is also a normal form under \mapsto_ν .

4.5. PROPOSITION. \mapsto_ν is strongly correct.

PROOF. It is very easy to check in Definition 4.3 that $A \mapsto_\nu B$ implies $A \sim B$. \square

The Propositions 4.4 and 4.5 tell us that \mapsto_ν^* defines a strongly correct normal form computation; we have only to ensure that it is the right one.

4.6. PROPOSITION. Let A be a formula and B a normal form of A with respect to \mapsto_ν . Then B is in negation normal form and $A \sim B$.

PROOF. $A \sim B$ directly follows from Proposition 4.5. Let us assume that B is in normal form with respect to \mapsto_ν , but B is not in NNF. Then, according to Definition 4.2 there are the following cases:

- (1) B contains a subformula of the form $F \rightarrow G$,
- (2) B contains a subformula of the form $F \leftrightarrow G$,
- (3) B contains a subformula of the form $\neg F$ where F is not an atom.

In case (1), B can be reduced by the rule nnf-1 , which contradicts the assumption that B is in \mapsto_ν -normal form. Similarly, in case (2), B can be reduced by nnf-2 , which again yields a contradiction. In case (3), there exists one of the rules $\text{nnf-3} - \text{nnf-7}$ which reduces B , again leading to a contradiction. \square

Clearly there are several, even infinitely many, formulae in NNF which are logically equivalent to a given formula A . The specific normal form(s) corresponding to A are defined by a reduction relation (like \mapsto_ν above). The relation \mapsto_ν is not only terminating, it is also confluent, i.e., arbitrary reduction sequences lead to the same normal form. Thus, relative to \mapsto_ν , we may speak about *the* NNF of a formula A . If we limit our input syntax to formulae not containing \leftrightarrow then the computation of NNF via \mapsto_ν can be performed in polynomial time; moreover, the normal form cannot be longer than double the size of the input formula. If we allow \leftrightarrow then, for $A \mapsto_\nu B$, the size of B is about double the size of A ; this may lead to an exponential expense. If we abandon strong correctness, we can find faster and more efficient methods based on extension by definition (see Section 6).

Another traditional first-order form is prenex normal form. Here, the aim consists in separating the quantifiers from a quantifier-free part, the so-called matrix. Unlike to NNF, the propositional structure of the formula remains unchanged.

4.7. EXAMPLE. Let A be the formula

$$(\exists x)((\forall y)(P(x, y) \wedge R(x)) \rightarrow (\forall u)P(u, u)).$$

Our aim is to shift all quantifiers in front, i.e., to the end, no quantifier should be in the scope of a connective. $(\exists x)$ is already at the right position, but $(\forall y)$ and $(\forall u)$ are not. In order to shift those quantifiers in front, we can make use of the quantifier shifting rules

$$((\forall v)F) \rightarrow G \mapsto (\exists v)(F \rightarrow G), \quad (4.1)$$

$$G \rightarrow (\forall v)F \mapsto (\forall v)(G \rightarrow F), \quad (4.2)$$

where we assume that G does not contain the variable v . Both rules above are applicable to A and we may choose some order of application. If we start with the first rule and apply the second one afterwards, we obtain

$$B : (\exists x)(\exists y)(\forall u)((P(x, y) \wedge R(x)) \rightarrow P(u, u)).$$

Applying the second rule prior to the first one gives

$$C : (\exists x)(\forall u)(\exists y)((P(x, y) \wedge R(x)) \rightarrow P(u, u)).$$

Both B and C are logically equivalent to A and in "prenex" form, i.e., all quantifiers in the formulae occur as a prefix.

Example 4.7 shows that, by using quantifier shifting rules, we may obtain different normal forms. Although they are logically equivalent, they may behave quite differently if further normalization steps (like skolemization) are applied; this problem area is treated in more detail in Section 5. In order to avoid renaming of bound variables during normalization, which might be necessary to the applicability of quantifier shifting, we may produce an adequate "version" of a first-order formula in advance.

4.8. DEFINITION (rectified formula). A first-order formula F is called *rectified* if the following property is fulfilled:

Let λ_1 and λ_2 be two different subformula occurrences in F , such that $(Q_1 v_1)G$ occurs at λ_1 and $(Q_2 v_2)H$ occurs at λ_2 ; then $v_1 \neq v_2$.

Roughly speaking, different quantifiers bind different variables.

Usually, we also have to postulate that all free variables are different from the bound ones, but this property is automatically fulfilled in our syntax definition of predicate logic. Clearly, every first-order formula A has a rectified form B and $A \sim B$; the preservation of logical equivalence directly follows from the definition of the semantics of first-order logic. We are now ready to give a general definition of prenex form and of a prenexing reduction relation.

4.9. DEFINITION (prenex form). A first-order formula F is in *prenex form* if it is rectified and no quantifier in F is in the scope of some connective, i.e., F is of the form

$$(Q_1 v_1) \dots (Q_n v_n)G,$$

where v_1, \dots, v_n are different variables and G is quantifier-free. G is called the *matrix* of F .

Note that the requirement of rectification in Definition 4.9 serves the purpose to exclude pathological formulae like $(\forall x)(\exists x)P(x, x)$. For the sake of simplicity (there is no quantifier shifting rule for \leftrightarrow), we define prenexing only on formulae not containing \leftrightarrow . In order to achieve such a form we have to normalize with respect to the rule

$$A \leftrightarrow B \mapsto (A \rightarrow B) \wedge (B \rightarrow A).$$

4.10. DEFINITION. Let \mapsto_π be the following reduction relation on rectified formulae not containing \leftrightarrow (Q stands for \forall and \exists , $\forall^d = \exists$ and $\exists^d = \forall$):

pnf-1 $\neg(Qv)A \mapsto_\pi (Q^d v)\neg A$,

pnf-2 $((Qv)A) \odot B \mapsto_\pi (Qv)(A \odot B)$ for $\odot \in \{\wedge, \vee\}$,

pnf-3 $A \odot (Qv)B \mapsto_\pi (Qv)(A \odot B)$ for $\odot \in \{\wedge, \vee\}$,

pnf-4 $((Qv)A) \rightarrow B \mapsto_\pi (Q^d v)(A \rightarrow B)$,

pnf-5 $A \rightarrow (Qv)B \mapsto_\pi (Qv)(A \rightarrow B)$.

The reduction by \mapsto_π can be applied on every subformula, i.e., we extend \mapsto_π in the following way: Let λ be a position in a formula F and G be the subformula occurring at λ and $G \mapsto_\pi H$. Then we define $F \mapsto_\pi F[H]_\lambda$.

Note that, for rectified A and $A \mapsto_\pi B$, B is rectified too, i.e., \mapsto_π is indeed a reduction relation on rectified formulae. Moreover, if we drop all quantifiers from A and B then they become equal; that means \mapsto_π leaves the propositional structure unchanged.

4.11. PROPOSITION. \mapsto_π is strongly correct.

PROOF. It is easy to verify (take any calculus of first-order logic) that $A \mapsto_\pi B$ implies $A \sim B$. \square

4.12. PROPOSITION. \mapsto_π is terminating.

PROOF. Like in Proposition 4.4, we have to define a finite measure on formulae which is decreased by \mapsto_π . Let F be a rectified formula without \leftrightarrow . For every occurrence λ of a connective $\wedge, \vee, \rightarrow, \neg$ in F , determine the number $q(\lambda)$ of quantifiers occurring below λ . Let $m(F)$ be the sum over all $q(\lambda)$ where λ ranges over all positions of connectives in F . Then, by definition of \mapsto_π via pnf-1 to pnf-5, $A \mapsto_\pi B$ implies $m(A) > m(B)$. Thus, the length of any reduction sequence is bounded by $m(A)$. \square

The two propositions above show that, for any rectified and \leftrightarrow -free formula A , any sequence of \mapsto_π reductions leads to a normal form B with $B \sim A$. It remains to show that B is indeed in prenex form.

4.13. PROPOSITION. Let A be a formula and B be a normal form of A with respect to \mapsto_π . Then B is in prenex normal form and $A \sim B$.

PROOF. If $A \mapsto_\pi B$ (on rectified \leftrightarrow -free formulae) then, due to the validity of the quantifier shifting rules $A \sim B$.

It remains to be shown that B , which is irreducible under \mapsto_π , is in PNF. Let us assume that B is not in prenex form, i.e., there exists an occurrence of a quantifier below a connective. Thus, there exists a subformula F of B occurring at a position λ having one of the following forms:

- (1) $F = (Qv)G \circ H$ for $\circ \in \{\wedge, \vee, \rightarrow\}$,
- (2) $F = G \circ (Qv)H$ for $\circ \in \{\wedge, \vee, \rightarrow\}$,
- (3) $F = \neg(Qv)G$.

In all cases (1), (2) and (3), there exists a rule pnf- i in Definition 4.10 which reduces F to F' and therefore $B \mapsto_\pi B[F']_\lambda$. But this contradicts the irreducibility of B . \square

The relation \mapsto_π does not allow to speak about *the* normal form of a formula. E.g., reconsider Example 4.7, where the formula

$$A : (\exists x)((\forall y)(P(x, y) \wedge R(x)) \rightarrow (\forall u)P(u, u))$$

has to be normalized. If we apply pnf-4 first and pnf-5 afterwards, the \mapsto_{π} -normal form is

$$B : (\exists x)(\exists y)(\forall u)((P(x, y) \wedge R(x)) \rightarrow P(u, u)).$$

In a reversed application of these rules, we obtain another \mapsto_{π} -normal form

$$C : (\exists x)(\forall u)(\exists y)((P(x, y) \wedge R(x)) \rightarrow P(u, u)).$$

Therefore, $A \mapsto_{\pi} B$ and $A \mapsto_{\pi} C$. Moreover, B and C are in normal form and $B \neq C$. This shows that the specific prenex form eventually obtained depends on the order of rule applications. Thus, different normal form operators yield different results. Note that this kind of ambiguity is by no means disturbing: logical normal forms should not be compared with results of arithmetical operations, where uniqueness is absolutely necessary! By choosing a fixed priority for the rules pnf-1 to pnf-5, we obtain a normal form operator which can be computed in quadratic time; the length of the resulting prenex form is the same as of the original formula. Though prenexing is algorithmically inexpensive, it may become very expensive in further use (as an input to a theorem prover). Effects of this type are analyzed in Section 5.

5. Skolem Normal Form

A simple and very important normalization is the elimination of *strong* quantifiers called *skolemization*. In case of refutational theorem proving and negation normal forms, the eliminated quantifiers are exactly the existential ones. Skolemization, unlike many other normalizations, does not only simplify the given formula but also extends its signature; in this sense, skolemization is also an *extension* method. Another particular feature is the loss of logical equivalence under this transformation; rather we only preserve sat-equivalence or validity-equivalence, respectively, in transforming a formula into Skolem normal form. In this section, our aim is to show that there is nothing like *the* skolemization of a formula and that different principles of quantifier elimination lead to formulae with substantially different properties. In particular, we will show that the range of quantifiers plays a crucial role in the proof complexities of Skolem normal forms.

5.1. EXAMPLE.

$$F : (\forall x)[(\exists y)P(x, y) \wedge (\exists z)Q(z)] \wedge (\forall u)(\neg P(a, u) \vee \neg Q(u)).$$

It is easy to see that F is satisfiable and nonvalid. Our aim is to transform F into a purely universal form by removing the existential quantifiers. We observe that $(\exists y)$ and $(\exists z)$ are dominated by $(\forall x)$; thus y and z "depend" on x . We express these dependencies by one-place function symbols, eliminate the existential quantifiers and replace the existential variables by new terms. The result is the formula

$$G : (\forall x)[P(x, f(x)) \wedge Q(g(x))] \wedge (\forall u)(\neg P(a, u) \vee \neg Q(u)).$$

G is satisfiable too, but not logically equivalent to F . Indeed, $G \rightarrow F$ is valid, but $F \rightarrow G$ is not. At least, we have $G \sim_{sat} F$.

It is very important that we replace y and z by functional terms with different function symbols; otherwise we get a formula

$$G' : (\forall x)[P(x, f(x)) \wedge Q(f(x))] \wedge (\forall u)(\neg P(a, u) \vee \neg Q(u)).$$

But G' is unsatisfiable (construct a resolution refutation out of the clauses $P(x, f(x))$, $Q(f(x))$, $\neg P(a, u) \vee \neg Q(u)$). Therefore, the transformation of F to G' is incorrect in the sense that $F \not\sim_{sat} G'$.

By carefully analyzing the structure of F , we notice that $(\exists z)$ is in the scope of $\forall x$; but this overbinding can be avoided by applying the quantifier shifting rule

$$(\forall v)(A(v) \wedge B) \rightarrow (\forall v)A(v) \wedge B$$

where $A(v)$ contains v but B does not.

The result then is

$$F_1 : (\forall x)(\exists y)P(x, y) \wedge (\exists z)Q(z) \wedge (\forall u)(\neg P(a, u) \vee \neg Q(u)).$$

In F_1 , $(\exists y)$ is in the scope of $(\forall x)$, but $(\exists z)$ is outside the scope of other quantifiers. Thus "z does not depend on x" and we replace z by a constant c symbolizing a true instance of $Q(\cdot)$. So we obtain

$$G_1 : (\forall x)P(x, f(x)) \wedge Q(c) \wedge (\forall u)(\neg P(a, u) \vee \neg Q(u)).$$

Again, we have $G_1 \sim_{sat} F_1$ and (by $F \sim F_1$) $F \sim_{sat} G_1$. Note that $G \sim_{sat} G_1$, but not $G \sim G_1$!

Instead of minimizing the scope of quantifiers, we may also decide for a prenex form, i.e., we shift all quantifiers in front before we eliminate the existential quantifiers. A prenex form (there are several others too) for F is

$$F_2 : (\forall x)(\forall u)(\exists y)(\exists z)[P(x, y) \wedge Q(z) \wedge (\neg P(a, u) \vee \neg Q(u))].$$

Now both existential quantifiers $(\exists y)$ and $(\exists z)$ are in the scope of $(\forall x)$, $(\forall u)$. Thus, their dependence is naturally expressed by two two-place function symbols h and i and we obtain

$$(\forall x)(\forall u)[P(x, h(x, u)) \wedge Q(i(x, u)) \wedge (\neg P(a, u) \vee \neg Q(u))].$$

Again, $G_2 \sim_{sat} F$ and also G_2 may serve as a \exists -free form for F . Note that G , G_1 and G_2 are mutually inequivalent.

By changing F to

$$H : (\forall x)((\exists y)P(x, y) \wedge (\exists z)Q(z)) \wedge (\forall u)(\forall v)(\neg P(a, u) \vee \neg Q(v)),$$

we obtain an unsatisfiable set of clauses. Then the corresponding \exists -free forms for H are unsatisfiable too; their clause forms can then be refuted by resolution. The \exists -free form corresponding to G is

$$G'' : (\forall x)(P(x, f(x)) \wedge Q(g(x))) \wedge (\forall u)(\forall v)(\neg P(a, u) \vee \neg Q(v))$$

and the corresponding (unsatisfiable) set \mathcal{C} of clauses is

$$\{P(x, f(x)), Q(g(x)), \neg P(a, u) \vee \neg Q(v)\}.$$

If we apply quantifier shifting again to minimize or maximize the range of quantifiers, we obtain different sets of clauses and different resolution refutations. In this very simple example, the corresponding proofs hardly differ. But we will see later that, in more complex examples, the effect can be dramatic.

Example 5.1 indicates that the *transformation* to a sat-equivalent formula is crucial and not the mere existence of such a formula. Let R be an arbitrary one-place predicate symbol: then clearly $F \sim_{\text{sat}} R(a) \vee \neg R(a)$ and $H \sim_{\text{sat}} R(a) \wedge \neg R(a)$ for F, H as in Example 5.1. Nevertheless, the forms $R(a) \vee \neg R(a)$ and $R(a) \wedge \neg R(a)$ are meaningless in a two-fold sense: (1) the satisfiability problem is undecidable and so there exists no computable transformation to formulae with a trivial test for satisfiability. (2) a refutation of $R(a) \wedge \neg R(a)$ has nothing to do with a refutation of H ; in fact, there is no reasonable way to construct a refutation of H out of the trivial refutation $R(a), \neg R(a), \square$. Moreover, G in Example 5.1 is not only sat-equivalent to F but even $G \rightarrow F$ is valid, while $(R(a) \vee \neg R(a)) \rightarrow F$ is not valid. The property $G \models F$ allows to reduce the problem to find a model for F to that for G . This property is important to the discipline of automated model building [Fermüller and Leitsch 1996].

In Example 5.1, we presented skolemization as a method eliminating existential quantifiers. In general, skolemization can be defined as a method eliminating strong quantifiers (see Section 2). We first define structural skolemization as an operator on closed, rectified first-order formulae (see Definition 4.8). The elimination of a strong quantifier (Qx) depends on the weak quantifiers having (Qx) in their scope.

From now on, we assume that all formulae are \leftrightarrow -free, e.g., \leftrightarrow can be eliminated by the rule

$$A \leftrightarrow B \mapsto (A \rightarrow B) \wedge (B \rightarrow A).$$

This restriction is necessary as, in presence of \leftrightarrow , subformulae do not have a unique polarity: as $(\forall x)B \leftrightarrow C \sim (\neg(\forall x)B \vee C) \wedge (\neg C \vee (\forall x)B)$, $(\forall x)B$ appears in negative *and* positive polarity, $(\forall x)$ once appears as strong and then as weak quantifier. This effect is excluded in formulae defined over $\{\wedge, \vee, \rightarrow, \neg, \forall, \exists\}$.

5.2. DEFINITION. Let A be a rectified formula without \leftrightarrow and $F : (Qx)G$ be a subformula of A . Then, for every subformula $(Q'y)H$ of G , we define $(Qx) <_A (Q'y)$; in this case, we say that $(Q'y)$ is in the scope of (Qx) in the formula A .

Indeed, if A is rectified, $(Qx) <_A (Q'y)$ and $(Q'y) <_A (Qx)$ cannot hold simultaneously. But $<_A$ is not a total ordering, because quantifiers may be incomparable (e.g., if $A = B \wedge C$ and (Qx) occurs in A and $(Q'y)$ occurs in B). However, we may extend $<_A$ to a total ordering \prec_A fulfilling: $(Qx) <_A (Q'y)$ implies $(Qx) \prec_A (Q'y)$. With respect to such an ordering \prec_A , we may speak about the *first strong quantifier*, i.e., a strong quantifier (Qx) such that all $(Q'y)$ with $(Q'y) \prec_A (Qx)$ are weak.

5.3. DEFINITION (*structural Skolem form*). Let A be a closed and rectified first-order formula. If A does not contain strong quantifiers, we define

$$sk(A) = A.$$

Suppose now that A contains strong quantifiers and (Qy) is the first strong quantifier occurring in A (with respect to a total ordering \prec_A corresponding to $<_A$). If (Qy) is not in the scope of weak quantifiers then

$$sk(A) = sk(A_{-(Qy)}\{y \leftarrow c\})$$

where $A_{-(Qy)}$ is the formula A after omission of (Qy) and c is a constant symbol not occurring in A . If (Qy) is in the scope of the weak quantifiers $(Q_1x_1) \dots (Q_nx_n)$ (where $(Q_1x_1) \prec_A \dots \prec_A (Q_nx_n)$) then

$$sk(A) = sk(A_{-(Qy)}\{y \leftarrow f(x_1, \dots, x_n)\})$$

where f is a function symbol not occurring in A . The formula $sk(A)$ is called *structural Skolem form* of A .

Note that the recursion in Definition 5.3 is well-defined (all formulae contain only finitely many quantifiers). Moreover, sk is a *function* of type $PL \rightarrow PL$; this justifies the term *the structural Skolem form*. It is easy to see that, besides the names of function symbols and constant symbols, $sk(A)$ only depends on $<_A$, but not on the specific extension \prec_A of $<_A$. Therefore, introduction of a nonterminism via $<_A$ does not change the normal form (essentially). In this definition strong quantifiers are eliminated "on the spot", i.e., the formula is not subject to any other transformation. By performing additional transformations (changing the relation $<_A$) prior to application of sk , we obtain other forms of skolemization which may differ in the functional term structure of the resulting formula.

5.4. DEFINITION. Let A be a closed first-order formula and A' be a prenex form of A ; then $sk(A')$ is called a *prenex Skolem form* of A . If A' is an antiprenex form of A (i.e., A' is obtained from A by minimizing the range of quantifiers by quantifier shifting rules) then $sk(A')$ is called an *antiprenex Skolem form* of A .

We will see in this section that the particular form of skolemization actually matters; it may strongly influence proof complexity and proof search—although the different normal forms are of about the same length.

Definition 5.3 gives us a method to transform every prenex form into an existential prenex form (i.e., the universal quantifiers are eliminated). This, however, does not correspond to Example 5.1 and to skolemization in refutational theorem proving in general; there, the \exists -quantifiers are those actually eliminated. In using a sequent notation, this problem is easily solved (*refutational* skolemization then takes place on the lefthandside of sequents).

5.5. DEFINITION. Now let $S : A_1, \dots, A_n \vdash B_1, \dots, B_m$ be a closed sequent and let $(A'_1 \wedge \dots \wedge A'_n) \rightarrow (B'_1 \vee \dots \vee B'_m)$ be the structural Skolem form of a rectified variant of the formula $(A_1 \wedge \dots \wedge A_n) \rightarrow (B_1 \vee \dots \vee B_m)$ (if $m = 0$ then the consequent of the implication is set to falsum; if $n = 0$ then the antecedent is set to verum). The sequent

$$S : A'_1, \dots, A'_n \vdash B'_1, \dots, B'_m$$

is called the (structural) *Skolem form* of S . If $A''_1, \dots, A''_n, B''_1, \dots, B''_m$ are prenex forms of $A_1, \dots, A_n, B_1, \dots, B_m$ then the structural Skolem form of

$$A''_1, \dots, A''_n \vdash B''_1, \dots, B''_m$$

is called a *prenex Skolem form* of S .

If $B \vdash$ is a Skolem form of $A \vdash$ then B is called a *refutational Skolem form* of A .

Note that a Skolem form $sk(S)$ of a sequent S is unique up to the renaming of (bound) variables (there might be different ways to rectify the implicational formula representing the sequent) and Skolem function symbols.

5.6. EXAMPLE. Let $A = (\forall x)(\exists y)P(x, y)$. Then $sk(A) = (\exists y)P(c, y)$ for a constant symbol c and $(\exists y)P(c, y)$ is the structural Skolem form of A . But $sk(A \vdash) = (\forall x)P(x, f(x)) \vdash$ and $(\forall x)P(x, f(x))$ is the (structural) refutational Skolem form of A .

The formula G in Example 5.1 is a structural refutational Skolem form of F and G_1 is an antiprenex refutational Skolem form of F . The formula

$$F' : ((\exists y)P(b, y) \wedge (\exists z)Q(z)) \wedge (\neg P(a, c) \vee \neg Q(c))$$

is the (ordinary) structural Skolem form of F .

We have seen in Example 5.1 that (refutational) skolemization cannot preserve logical equivalence but only sat-equivalence. More generally, skolemization of sequents preserves *validity*-equivalence:

5.7. THEOREM. A closed sequent S is valid iff $sk(S)$ is valid.

PROOF. Let $S : A_1, \dots, A_m \vdash B_1, \dots, B_n$ then S is valid iff

$$F(S) : (A_1 \wedge \dots \wedge A_m) \rightarrow (B_1 \vee \dots \vee B_n)$$

is valid. By Definition 5.5, the skolemization of S is defined via that of $F(S)$. Therefore, it suffices to show:

(*) A closed rectified formula B is valid iff $sk(B)$ is valid, i.e., $B \sim_{val} sk(B)$.

We proceed by induction on the number $\delta(B)$ of occurrences of strong quantifiers in B .

$\delta(B) = 0$: $B \sim_{val} sk(B)$ by $B = sk(B)$.

(IH) Let us assume that (*) holds for all B with $\delta(B) \leq n$.

In the case $\delta(B) = n + 1$, select the first strong quantifier (Qx) occurring in B (corresponding to some ordering \prec_B); such a quantifier exists as $\delta(B) > 0$. We distinguish two subcases:

Subcase a: (Qx) is not in the scope of other quantifiers in B .

Then, by Definition 5.3,

$$sk(B) = sk(B_{-(Qx)}\{x \leftarrow c\})$$

for a constant c not occurring in B .

By (IH), $B_{-(Qx)}\{x \leftarrow c\} \sim_{val} sk(B_{-(Qx)}\{x \leftarrow c\})$ and it is sufficient to show

$$B \sim_{val} B_{-(Qx)}\{x \leftarrow c\}.$$

As B is rectified and (Qx) is strong, we have $B \sim B' : (\forall x)B_{-(Qx)}$ (this form can be obtained by repeated applications of quantifier-shifting rules). Trivially, the formula $B' \rightarrow B_{-(Qx)}\{x \leftarrow c\}$ is valid; thus, if B' is valid then $B_{-(Qx)}\{x \leftarrow c\}$ is valid too.

For the other direction assume that B' is not valid, i.e., $\text{val}_\Gamma(B') = \text{false}$ for some interpretation Γ . By the definition of first-order interpretations, there exists an interpretation Γ' which is equivalent to Γ modulo x (i.e., Γ and Γ' differ at most in the interpretation of the variable x) and $\text{val}_{\Gamma'}(B_{-(Qx)}) = \text{false}$. But then, there also exists an interpretation Γ'' of $B_{-(Qx)}\{x \leftarrow c\}$ where c gets the value of x in Γ' and Γ'' coincides with Γ' otherwise (note that c does not occur in B !). This yields

$$\text{false} = \text{val}_{\Gamma'}(B_{-(Qx)}) = \text{val}_{\Gamma''}(B_{-(Qx)}\{x \leftarrow c\})$$

and, consequently, $B_{-(Qx)}\{x \leftarrow c\}$ is not valid. Therefore, $B' \sim_{val} B_{-(Qx)}\{x \leftarrow c\}$ and, as $B' \sim B$, also $B \sim_{val} B_{-(Qx)}\{x \leftarrow c\}$.

Subcase b: (Qx) is in the scope of the weak quantifiers $(Q_1y_1) \dots (Q_ny_n)$.

As weak quantifiers become existential when they are shifted in front, we get

$$B \sim B' : (\exists y_1) \dots (\exists y_n)(\forall x)C,$$

where C is the formula B without the quantifiers $(\exists y_1) \dots (\exists y_n), (\forall x)$. By Definition 5.3,

$$sk(B') = sk((\exists y_1) \dots (\exists y_n)C\{x \leftarrow f(y_1, \dots, y_n)\}),$$

where f is a function symbol which does not occur in B . After application of the induction hypothesis, we are left with the problem

$$B' \sim_{val} B'' : (\exists y_1) \dots (\exists y_n)C\{x \leftarrow f(y_1, \dots, y_n)\}.$$

Clearly, $B' \rightarrow B''$ is valid and thus the validity of B' implies that of B'' .

For the other direction, let us assume that B' is not valid, i.e., there exists an interpretation Γ such that $\text{val}_\Gamma(B') = \text{false}$. Then clearly $\text{val}_\Gamma(\neg B') = \text{true}$, i.e., Γ is a model of

$$(\forall y_1) \dots (\forall y_n)(\exists x)\neg C.$$

By the semantics of first-order logic, this implies that, for all interpretations of y_1, \dots, y_n , there exist an interpretation of x such that $\neg C$ evaluates to true. By the axiom of choice, there exists a function φ assigning such an interpretation of x to every interpretation of the y_1, \dots, y_n . Now we extend the interpretation Γ to an interpretation Γ^* by interpreting the function symbol f as φ ; note that there can be no conflict as f does not occur in B and B' . Then Γ^* is a model of

$$(\forall y_1) \dots (\forall y_n)\neg C\{x \leftarrow f(y_1, \dots, y_n)\}$$

and so Γ^* falsifies B'' , which proves that B'' is not valid. Thus, we have shown $B' \sim_{\text{val}} B''$. Now it is easy to see that $B'' \sim B_{-(Qx)}\{x \leftarrow f(y_1, \dots, y_n)\}$ —implying $sk(B') \sim sk(B)$ —and therefore $B \sim_{\text{val}} sk(B)$. \square

5.8. COROLLARY. *Let A be a closed formula and $skr(A)$ be the refutational (structural) Skolem form of A . Then $A \sim_{\text{sat}} skr(A)$.*

PROOF. Observe that $sk(A \vdash) = B \vdash$ iff $skr(A) = B$. By the semantics of sequents, $B \vdash$ is valid iff B is unsatisfiable; similarly, $A \vdash$ is valid iff A is unsatisfiable. By Theorem 5.7, $A \vdash \sim_{\text{val}} B \vdash$; and, consequently, A is unsatisfiable iff B is unsatisfiable. \square

There are more methods of skolemization than just structural, prenex and antiprenex one. In Andrews' method (defined and proved correct in [Andrews 1971] and [Andrews 1981]), the introduced skolem functions do not depend on the weak quantifiers $(Q_1x_1), \dots, (Q_nx_n)$ dominating the strong quantifier (Qx) , but rather on the subset of $\{x_1, \dots, x_n\}$ appearing (free) in the subformula dominated by (Qx) . Similarly to antiprenex skolemization, this method leads to smaller Skolem terms in general.

5.9. DEFINITION (*Andrews' Skolem form*). Let A be a closed and rectified first-order formula. If A does not contain strong quantifiers, we define

$$sk_A(A) = A.$$

Suppose now that A contains strong quantifiers, $(Qy)B$ is a subformula of A , and (Qy) is the first strong quantifier occurring in A (with respect to a total ordering \prec_A corresponding to $<_A$). If $(Qy)B$ has no free variables which are weakly quantified then

$$sk_A(A) = sk_A(A_{-(Qy)}\{y \leftarrow c\}),$$

where $A_{-(Qy)}$ is the formula A after omission of (Qy) and c is a constant symbol not occurring in A . If $(Qy)B$ has n free variables x_1, \dots, x_n which are weakly quantified, then

$$sk_A(A) = sk_A(A_{-(Qy)}\{y \leftarrow f(x_1, \dots, x_n)\}),$$

where f is a function symbol not occurring in A . The formula $sk_A(A)$ is called *Andrews' Skolem form* of A . The refutational variant, denoted by $skr_A(A)$ is defined *mutatis mutandis*.

5.10. EXAMPLE. Let $S : (\forall z)(\forall y)((\exists x)P(y, x) \vee Q(y, z)) \vdash$. The structural Skolem form of S is

$$S_1 : (\forall z)(\forall y)(P(y, f(z, y)) \vee Q(y, z)) \vdash.$$

As the quantifier $(\forall y)$ cannot be shifted inwards by (only) using the shifting rules, the antiprenex Skolem form of S is also S_1 . But the subformula $(\exists x)P(y, x)$ does not contain the variable z ; thus, the Skolem form obtained via Andrews' method is

$$S_3 : (\forall z)(\forall y)(P(y, g(y)) \vee Q(y, z)) \vdash.$$

Note that a one-place Skolem symbol can also be obtained by a logically equivalent transformation followed by structural skolemization.

Transform S into

$$S' : (\forall y)(\forall z)((\exists x)P(y, x) \vee Q(y, z)) \vdash$$

and S' via a quantifier shifting rule into

$$S'' : (\forall y)((\exists x)P(y, x) \vee (\forall z)Q(y, z)) \vdash.$$

The structural skolemization of S'' then gives

$$S^* : (\forall y)(P(y, g(y)) \vee (\forall z)Q(y, z)) \vdash.$$

Although S^* is not equal to S_3 , they are logically equivalent (via the same logical rules applied for the construction of S'').

On the other hand, antiprenex skolemization is not "weaker" than Andrews' one (in the sense of reducing the arity of Skolem terms). Just take the prenex form

$$S_0 : (\forall y)(\forall z)(\exists x)(P(y, x) \vee Q(y, z)) \vdash$$

Then both structural and Andrews' skolemization introduce a two-place function symbol, while antiprenex skolemization introduces a one-place function symbol.

A much more sophisticated method was developed by Goubault [1995]; it consists of a clever range reduction of quantifiers using binary decision diagrams of the formulae prior to skolemization. In restricting the arity of Skolem symbols, Goubault's method surpasses the other methods mentioned above. The presentation of his method, however, is beyond the scope of this article.

In general, we may define a Skolem form of a sequent S by a two-step procedure of the following type:

Step 1. Transform S into a sequent S' such that $S \sim S'$.

Step 2. Compute $sk(S')$.

Clearly, such a transformation makes sense in practice only if the transformation in Step 1. is reasonably fast (e.g., computable in polynomial time). This condition is clearly fulfilled by antiprenexing. The question remains whether it is relevant at all to investigate different methods of skolemization. Comparing prenex, structural and antiprenex skolemization, we see that the lengths of the forms do not differ substantially; in particular, all of them are polynomial in the length of the original sequent. On the surface, higher arities of Skolem symbols look ugly but not dangerous to automated deduction. This picture, however, changes when we focus on proof complexity instead on length of representations. In fact, an inadequate method of skolemization may destroy logical *information* stored in a formula making proofs dramatically longer. This effect is largely independent of the specific inference system and can conveniently be described in terms of Herbrand complexity. The complexity analysis is based on several complexity measures which are defined below.

5.11. DEFINITION. The logical complexity, $comp(A)$, of a formula A is defined as the number of occurrences of logical operators in A . Formally, we define

- $comp(A) = 0$ for atom formulae A ,
- $comp(A \circ B) = 1 + comp(A) + comp(B)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$,
- $comp(\neg A) = 1 + comp(A)$,
- $comp((Qx)A) = 1 + comp(A)$ for $Q \in \{\forall, \exists\}$.

5.12. DEFINITION. Let $S : A_1, \dots, A_n \vdash B_1, \dots, B_m$ be a sequent. Then we define the length of S as $||S|| = n + m$ (i.e., the number of formula occurrences in S).

5.13. DEFINITION. The *size* of a formula F , denoted by $|F|$, is the number of symbol occurrences in the string representation of the formula. If $\Delta = F_1, \dots, F_n$ or $\Delta = \{F_1, \dots, F_n\}$, then $|\Delta| = \sum_{i=1}^n |F_i|$. For a sequent $S = \Delta \vdash G$, $|S| = |\Delta| + |G|$. The length $l(\alpha)$ of an LK-proof α is defined as the number of sequent occurrences in α (this corresponds to the number of nodes in the proof tree). The size of an LK-proof α , denoted by $|\alpha|$, is the sum of the sizes of all sequent occurrences in α . The size of a (tree) resolution derivation Φ , denoted by $|\Phi|$, is the sum of the sizes of all clause occurrences in Φ .

5.14. DEFINITION (*Herbrand complexity*). Let $S : A_1, \dots, A_n \vdash B_1, \dots, B_m$ be a valid sequent containing weak quantifiers only (i.e., $sk(S) = S$). Let $S^0 : A_1^0, \dots, A_n^0 \vdash B_1^0, \dots, B_m^0$ be a rectified variant of S . Furthermore, let $A'_1, \dots, A'_n, B'_1, \dots, B'_m$ be the sequent formulae of S^0 without quantifiers and let $A'_{j,1}, \dots, A'_{j,r_j}$ be substitution instances of the A'_j and similarly $B'_{j,1}, \dots, B'_{j,s_j}$ of the B'_j . A valid sequent of the form

$$A'_{1,1}, \dots, A'_{1,r_1}, \dots, A'_{n,1}, \dots, A'_{n,r_n} \vdash B'_{1,1}, \dots, B'_{1,s_1}, \dots, B'_{m,1}, \dots, B'_{m,s_m}$$

is called a *Herbrand sequent* of S .

The number

$$\mathcal{HC}(S) = \min\{\|S'\| \mid S' \text{ is a Herbrand sequent of } S\}$$

is called the *Herbrand complexity* of S .

5.15. EXAMPLE. Let S be the sequent $P(a), (\forall x)(P(x) \rightarrow P(f(x))) \vdash P(f(f(a)))$. Then the sequent

$$S' : P(a), P(a) \rightarrow P(f(a)), P(f(a)) \rightarrow P(f(f(a))) \vdash P(f(f(a)))$$

is a Herbrand sequent of S , $\|S'\| = 4$ and thus $\mathcal{HC}(S) \leq 4$. It is easy to verify that S' is the smallest Herbrand sequent possible and so $\mathcal{HC}(S) = 4$.

By Herbrand's theorem, every valid sequent containing weak quantifiers only defines a corresponding Herbrand sequent; thus, Herbrand complexity is well-defined. Still, \mathcal{HC} can be defined in several ways—depending on our choice of a complexity measure for sequents (an alternative would be to define $\|S\|$ as the number of symbol occurrences in the sequent S). As skolemization is a method of removing strong quantifiers, any sequent resulting from a valid one has a Herbrand complexity. The following theorem shows that different forms of skolemization may yield sequents with strongly different Herbrand complexity.

5.16. THEOREM. *There exists a constant $c \in \mathbb{N}$ and a sequence of sequents S_n such that*

- a) $\mathcal{HC}(S'_n) \leq 2^{2^{2^{cn}}}$ for the structural Skolem forms S'_n of S_n and $n \in \mathbb{N}$.
- b) *There exist prenex Skolem forms S''_n of S_n such that*

$$\mathcal{HC}(S''_n) \geq \frac{1}{2}s(n)$$

for the function s defined in Definition 2.16.

PROOF. We only give a proof sketch. The full proof can be found in [Baaz and Leitsch 1994] (Theorem 4.1).

By a famous result of Statman [1979], Herbrand complexity may be "nonelementarily" high in terms of proof complexity:

(*) There exists a sequence of formulae $(F_n)_{n \in \mathbb{N}}$ such that

- (1) $\mathcal{HC}(\vdash F_n) > \frac{1}{2}s(n)$ for all $n \geq 1$ and
- (2) there are LK-proofs φ_n of $\vdash F_n$ with $l(\varphi_n) \leq cn$ for $n \geq 1$ and an appropriate constant c (see [Statman 1979] and [Baaz and Leitsch 1994]).

The proofs φ_n contain cuts with cut formulae A_1^n, \dots, A_{2n+1}^n ; the logical complexity of the cut formulae is $< 2^{n+3}$.

The proofs φ_n can be transformed into simple proofs ψ_n of the same end sequents. LK-proofs are called simple if they are weakening-free and the initial sequents are atomic. As the proofs are represented in the form of trees, the worst case of this transformation is double exponential (the bound can be improved to simple exponential growth by a result in [Baaz and Leitsch 1999]). We thus obtain

(I) There exists a sequence of simple LK-proofs ψ_n of $\vdash F_n$ with closed cut formulae A_1^n, \dots, A_{2n+1}^n and $l(\psi_n) \leq cn2^{2n+3}$.

In the next step, we "eliminate" the cuts by replacing the cut rule by implication introduction to the left. This transformation is correct because the cut formulae are closed and, consequently, no eigenvariable condition can be violated. Thus every inference of the form

$$\frac{\Gamma \vdash \Delta, A \quad A, \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'} \text{ cut}$$

is replaced by

$$\frac{\Gamma \vdash \Delta, A \quad A, \Gamma' \vdash \Delta'}{A \rightarrow A, \Gamma, \Gamma' \vdash \Delta, \Delta'} \rightarrow: l$$

This transformation, which does not increase the proof length, is applied to all cuts in ψ_n ; the resulting proofs χ_n are simple cut-free proofs of sequents

(II) $S_n : A_1^n \rightarrow A_1^n, \dots, A_{2n+1}^n \rightarrow A_{2n+1}^n \vdash F_n$
and $l(\chi_n) \leq cn2^{2n+3}$.

The next step consists of skolemizing the proofs χ_n : the proofs χ_n are transformed into proofs χ'_n of the structural Skolem forms S'_n of the S_n . The new proofs are also simple, cut-free and $l(\chi'_n) \leq l(\chi_n)$. This transformation, which works for all simple cut-free proofs, consists of dropping the introduction of strong quantifiers and inserting appropriate Skolem terms. The sequents S'_n then are of the form

(III) $B_1^n \rightarrow C_1^n, \dots, B_{2n+1}^n \rightarrow C_{2n+1}^n \vdash F_n$.

The formulae F_n of Statman's sequence do not contain strong quantifiers and thus appear unchanged in S'_n . Note that, by the dual polarity of positions, the Skolem form of a formula $A \rightarrow A$ containing strong quantifiers is $B \rightarrow C$ for different formulae B and C . In [Baaz and Leitsch 1994], it is shown that, from every simple cut-free proof of a skolemized sequent, a Herbrand sequent can be extracted under at most exponential expense (note that the midsequent theorem [Takeuti 1975] cannot be applied as the sequents S'_n need not be in prenex form). This gives

(IV) $\mathcal{HC}(S'_n) \leq 2^{2^{cn}}$ for $n \geq 1$ and an appropriate constant c .

On the other hand, it is possible to construct prenex Skolem forms S''_n of S_n such that, for all $n \geq 1$,

(V) $\mathcal{HC}(S''_n) \geq \mathcal{HC}(\vdash F_n)$.

(V) expresses the fact that transforming sequents into prenex form may destroy important logical information stored in the formula to be proved. In our case, the effect of cuts is totally annihilated by prenexing and the formulae D_i in the sequents

$$S''_n : D_1, \dots, D_{2n+1} \vdash F_n$$

are completely redundant.

By (*), $\mathcal{HC}(\vdash F_n) > \frac{1}{2}s(n)$ and so $\mathcal{HC}(S''_n) > \frac{1}{2}s(n)$. □

As the function s is nonelementary, a direct consequence of Theorem 5.16 is the nonexistence of an elementary function f such that $\mathcal{HC}(S''_n) \leq f(\mathcal{HC}(S'_n))$ for

$n \in \mathbb{N}$; this result also holds if we define \mathcal{HC} via the number of symbol occurrences (instead of the number of formula occurrences) in a sequent. In [Baaz and Leitsch 1994], it is also shown that minimizing the range of quantifiers (via shifting rules) never leads to an increase of \mathcal{HC} ; on the contrary, if a corresponding sequence of prenex sequents is given in advance, antiprenex skolemization may give a nonelementary speed-up over the structural one! Thus, we see that different forms of skolemization may lead to very different sets of proofs. As a consequence of this result, we should consider skolemization as an integral part of the *inference* process and not as a mere "preprocessing" of minor importance.

6. Conjunctive Normal Form

Many calculi in automated deduction rely on a specific normal form, namely *clausal normal form*. There are two different possibilities what a clausal form (or synonymously, a set of sets of literals) can be: either a *disjunctive normal form* (DNF) of an input formula F , or a *conjunctive normal form* (CNF) of $\neg F$. In the former case, positive occurrences of \forall and negative occurrences of \exists are the strong quantifiers, whereas in the latter case, negative occurrences of \forall and positive occurrences of \exists are the strong quantifiers. As a consequence, both related transformations introduce Skolem functions at the same position in F either by affirmative skolemization or by refutational skolemization (see Section 5 for a detailed discussion on different skolemization principles). The only differences between a DNF of F and a CNF of $\neg F$ are the following: (i) the connectives \wedge and \vee are replaced by their duals \vee and \wedge , respectively, and (ii) any literal occurrence is replaced by its dual literal. Observe that any potential resolution inference is preserved by converting a DNF of F into a CNF of $\neg F$ and vice versa. As a consequence, it is not necessary (from a logical point of view) to favor one normal form over the other, but, since many calculi like resolution are usually explained as *refutational calculi*, a CNF is preferred for presentational reasons.

We first introduce two different kinds of transformations into normal form, namely *nonstructural* and *structural* transformations. Characteristic of nonstructural transformations is the loss of the structural information of the input formula. Subformulae are disrupted and their parts are distributed over different clauses of the resulting normal form. The main reason for the disruption is the application of distributivity laws in the transformation of the Skolem normal form into clauses. On the other hand, the syntactic structure is preserved by structural transformations which introduce an abbreviation (or a *label*) for any subformula occurrence of the input formula. We will introduce two variants of structural transformations and compare nonstructural with structural transformations and both variants of the structural transformations. We conclude this section with a brief discussion of practical merits of structural transformations.

There are many variants of nonstructural transformations into a CNF in the literature. Since we neither want to be concerned with different nonstructural transformations nor want to choose one specific variant, we follow [Baaz, Fermüller and

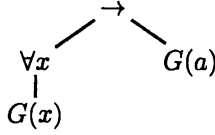


Figure 1: The formula tree for Example 6.2.

Leitsch 1994] and use an abstract characterization which is fulfilled by all these variants.

6.1. DEFINITION. A *nonstructural* transformation into clausal normal form is any mapping γ_{NS} from first-order sentences into sets of clauses that fulfills the following conditions.

1. If the closed input formula F is of the form

$$\neg(Q_1x_1) \dots (Q_mx_m)(F_1 \wedge \dots \wedge F_n \rightarrow G)$$

with $Q_i \in \{\forall, \exists\}$ ($1 \leq i \leq m$) then

$$\gamma_{NS}(F) = \gamma_{NS}((Q_1^d x_1) \dots (Q_m^d x_m)(F_1 \wedge \dots \wedge F_n \wedge \neg G)).$$

2. If L_1, \dots, L_n and M_1, \dots, M_m are literals then

$$\gamma_{NS}(\forall(L_1 \wedge \dots \wedge L_n \rightarrow M_1 \vee \dots \vee M_m)) = \{L_1^d, \dots, L_n^d, M_1, \dots, M_m\}.$$

3. $\gamma_{NS}(A_1 \wedge A_2) = \gamma_{NS}(A_1) \wedge \gamma_{NS}(A_2)$ for all sentences A_1 and A_2 .

4. If P is an atom and F any formula then, for all $C \in \gamma_{NS}(\forall(P \vee \neg P \vee F))$, both literals P and $\neg P$ occur in C .¹

$\gamma_{NS}(F)$ is called a *nonstructural normal form* of F .

In contrast to the corresponding definition in [Baaz et al. 1994], we use the additional condition 1. Observe that there is no assumption about the removal of strong quantifiers. Indeed, as we know from Section 5, there are several skolemization algorithms which can be used for this purpose. In the following, we use Andrews' skolemization technique where the Skolem functions introduced for a strong quantifier $(Qx)B$ depend on the free variables of $(Qx)B$ which are weakly quantified (cf. Definition 5.9).

Structural transformations are based on the introduction of labels for subformulae. Let us start with an example.

6.2. EXAMPLE. Let F be the formula $((\forall x)G(x)) \rightarrow G(a)$ and consider Figure 1 for the corresponding formula tree. We introduce labels $L_{G(a)}$, $L_{G(x)}(x)$, L_{\forall} , and L_F for the subformula occurrences $G(a)$, $G(x)$, $(\forall x)G(x)$, and F , respectively. Any

¹Observe that $\gamma_{NS}(\forall(P \vee \neg P \vee F))$ may be empty by deleting tautological clauses.

label must be a globally new predicate symbol with arity equal to the number of free variables in the corresponding subformula occurrence. The labels define the corresponding subformula occurrences, i.e., the (universal closures of the) following definitions are introduced.

$$\begin{array}{llll} L_{G(a)} & \leftrightarrow & G(a) & L_{G(x)}(x) \leftrightarrow G(x) \\ L_{\forall} & \leftrightarrow & (\forall x)L_{G(x)}(x) & L_F \leftrightarrow (L_{\forall} \rightarrow L_{G(a)}) \end{array}$$

Instead of translating $\neg F$ into CNF, the four definitions are translated into clause form using a nonstructural transformation. Observe that the equivalences can be easily transformed into clauses because they are flat (no deeply nested subformula can occur because any subformula occurrence is encoded by a label). Andrews' variant of refutational skolemization (i.e., the Skolem terms depend on the *free* variables of the quantified subformula) is applied for removing strong quantifiers. In summary, the following clauses are obtained where b is a new Skolem function symbol.

$$\begin{array}{ll} C_1 : \neg G(a) \vee L_{G(a)} & C_2 : \neg L_{G(a)} \vee G(a) \\ C_3 : \neg G(x) \vee L_{G(x)}(x) & C_4 : \neg L_{G(x)}(x) \vee G(x) \\ C_5 : \neg L_{G(x)}(b) \vee L_{\forall} & C_6 : \neg L_{\forall} \vee L_{G(x)}(x) \\ C_7 : \neg L_{G(a)} \vee L_F & C_8 : L_{\forall} \vee L_F \\ C_9 : \neg L_F \vee \neg L_{\forall} \vee L_{G(a)} & \end{array}$$

Moreover, $C_{10} : \neg L_F$ is added to the clause set obtained so far.

The example may suggest that a transformation of F based on the introduction of definitions leads to a normal form which is essentially longer than the corresponding nonstructural normal form of $\neg F$. This suggestion, however, is misleading in general. Here, our formula $\neg F$ is essentially in CNF (up to the replacement of \rightarrow by its definition, pushing negation inward and skolemization). In general, nonstructural normal forms can be exponential in the size of the input formula (even in the propositional case)—a disadvantage which does not occur for structural transformations. We will return to this point below.

A further drawback of nonstructural normal forms has been already mentioned: the loss of structural information of the input formula in the resulting normal form. This is of less importance, if we expect either Yes or No as the theorem prover's answer, but becomes very important in an interactive setting or if we want to get a readable proof of the input formula, which can be used for further tasks.

6.3. DEFINITION. Let F be a first-order formula. Then $\Sigma(F)$ denotes the set of all occurrences of subformulae of F which occur in its formula tree. Moreover, $\Sigma^+(F)$ denotes the set of all occurrences of subformulae with positive polarity in F and $\Sigma^-(F)$ denotes the set of all occurrences of subformulae with negative polarity in F .

G	C_G^+ and C_G^-
atomic	$C_G^+ : (\forall \vec{x}) (\neg G \vee L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (\neg L_G(\vec{x}) \vee G)$
$\neg M$	$C_G^+ : (\forall \vec{x}) (L_M(\vec{x}) \vee L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (\neg L_G(\vec{x}) \vee \neg L_M(\vec{x}))$
$H \wedge I$	$C_G^+ : (\forall \vec{x}) (\neg L_H(\vec{y}) \vee \neg L_I(\vec{z}) \vee L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (\neg L_G(\vec{x}) \vee L_H(\vec{y})) \wedge (\forall \vec{x}) (\neg L_G(\vec{x}) \vee L_I(\vec{z}))$
$H \vee I$	$C_G^+ : (\forall \vec{x}) (\neg L_H(\vec{y}) \vee L_G(\vec{x})) \wedge (\forall \vec{x}) (\neg L_I(\vec{z}) \vee L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (\neg L_G(\vec{x}) \vee L_H(\vec{y}) \vee L_I(\vec{z}))$
$H \rightarrow I$	$C_G^+ : (\forall \vec{x}) (L_H(\vec{y}) \vee L_G(\vec{x})) \wedge (\forall \vec{x}) (\neg L_I(\vec{z}) \vee L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (\neg L_G(\vec{x}) \vee \neg L_H(\vec{y}) \vee L_I(\vec{z}))$
$H \leftrightarrow I$	$C_G^+ : (\forall \vec{x}) (L_G(\vec{x}) \vee L_H(\vec{y}) \vee L_I(\vec{z})) \wedge$ $(\forall \vec{x}) (L_G(\vec{x}) \vee \neg L_H(\vec{y}) \vee \neg L_I(\vec{z}))$ $C_G^- : (\forall \vec{x}) (\neg L_G(\vec{x}) \vee L_H(\vec{y}) \vee \neg L_I(\vec{z})) \wedge$ $(\forall \vec{x}) (\neg L_G(\vec{x}) \vee \neg L_H(\vec{y}) \vee L_I(\vec{z}))$
$(\forall x) K$	$C_G^+ : (\forall \vec{x}) (\neg L_K(\vec{x}, g(\vec{x})) \vee L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (\forall x) (\neg L_G(\vec{x}) \vee L_K(\vec{x}, x))$
$(\exists x) K$	$C_G^+ : (\forall \vec{x}) (\forall x) (\neg L_K(\vec{x}, x) \vee L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (\neg L_G(\vec{x}) \vee L_K(\vec{x}, g(\vec{x})))$

Figure 2: The transformation rules for classical logic.

Observe that $\Sigma^+(F)$ and $\Sigma^-(F)$ are not disjoint in general. Subformula occurrences dominated by \leftrightarrow occur positively as well as negatively in F by definition.

6.4. DEFINITION. Let G be an occurrence of a first-order formula and let $\vec{x} = x_1, \dots, x_k$ be the free variables of G (without duplicates). The atom $L_G(\vec{x})$ is called an abbreviation (or *label*) for G .

6.5. DEFINITION. Let F be a closed first-order formula. For any $G \in \Sigma(F)$ with free variables $\vec{x} = x_1, \dots, x_k$, a label for G is introduced. Let $\vec{y} = y_1, \dots, y_l$ be the free variables of H , $\vec{z} = z_1, \dots, z_m$ be the free variables of I , where $\{\vec{y}\} \subseteq \{\vec{x}\}$, $\{\vec{z}\} \subseteq \{\vec{x}\}$, and $\{\vec{y}\} \cup \{\vec{z}\} = \{\vec{x}\}$. Moreover, \vec{x}, x are the free variables of K and \vec{x} are the free variables of M . The Skolem function symbol g is a globally new function symbol neither occurring in F nor being introduced in the transformation of any other subformula of F . The transformation rules are presented in Figure 2 (to be discussed below). The *definitional form* of F is the formula

$$\delta(F) = \bigwedge_{G \in \Sigma(F)} (C_G^+ \wedge C_G^-).$$

The *p-definitional form* (the definitional form obeying polarities) is the formula

$$\delta_p(F) = \left(\bigwedge_{G \in \Sigma^+(F)} C_G^+ \right) \wedge \left(\bigwedge_{G \in \Sigma^-(F)} C_G^- \right).$$

Let us come back to the transformation rules in Figure 2. We explain the case $G = (H \leftrightarrow I)$ in the following; the other cases are similar. Recall that the basic idea underlying the transformation into (p-)definitional form is the introduction of a new label for any subformula occurrence of the given input formula. Suppose that we have already translated H and I where $L_H(\vec{y})$ is the label for the subformula occurrence H and $L_I(\vec{z})$ is the label for the subformula occurrence I (the free variables are as described in the definition). For the subformula occurrence G , a label of the form $L_G(\vec{x})$ is introduced together with a closed equivalence of the form

$$(\forall \vec{x}) (L_G(\vec{x}) \leftrightarrow (L_H(\vec{y}) \leftrightarrow L_I(\vec{z}))). \quad (6.1)$$

In other words, we set the label for G equivalent to the considered subformula $H \leftrightarrow I$, but we replace H and I by their labels. A nonstructural transformation is applied to (6.1). First, the topmost equivalence is replaced by

$$(\forall \vec{x}) ((L_H(\vec{y}) \leftrightarrow L_I(\vec{z})) \rightarrow L_G(\vec{x})) \quad (6.2)$$

$$(\forall \vec{x}) (L_G(\vec{x}) \rightarrow (L_H(\vec{y}) \leftrightarrow L_I(\vec{z}))). \quad (6.3)$$

Both implications are processed independently; (6.2) yields C_G^+ and (6.3) yields C_G^- .

Since the term *structural transformation* is used for a variety of translations in the literature, we instead use the term (p-)definitional transformation for structure-preserving transformations into (p-)definitional form.

Historical Remarks. Although the introduction of new names for concepts by definitions is a well-known principle in logic and mathematics, similar principles were investigated remarkably late in the field of automated deduction. Tseitin [1968] was the first who used an *extension principle* for a simulation of proofs in the sequent calculus by the resolution calculus. While a cut-free proof of a formula F can be translated using $\delta(F)$ (even $\delta_p(F)$ suffices), the simulation of the cut rule requires the additional transformation of the cut formula by $\delta(\cdot)$. Hence, a transformation into definitional form is a kind of "analytic extension principle".

Cook and Reckhow [1974, 1979] used a transformation into definitional form (they called it *limited extension*) in transformations of a proof in one calculus into a corresponding proof of another calculus. The interesting question was whether the length of both proofs are polynomially related. Such an investigation is driven by the question whether there exists a "super proof system", i.e., a proof system in which *every* propositional tautology has a short proof (polynomial in the size of the input formula). The existence of a super proof system is closely related to a major question in complexity theory: there exists a super proof system iff \mathcal{NP} equals $\text{co-}\mathcal{NP}$.

Eder [1992] extended Tseitin's extension principles to the first-order case and proved various polynomial simulation results for first-order calculi. He investigated

(in [Eder 1984]) a structural transformation to normal form what we call $\delta(\cdot)$. Independently, Plaisted and Greenbaum [1986] proposed another structural transformation which essentially coincides with $\delta_p(\cdot)$.

Boy de la Tour [1990, 1992] combined definitional and nonstructural translations in order to get smaller resulting normal forms. In [Nonnengart, Rock and Weidenbach 1998], this approach was further refined and extended by a skolemization procedures which produces clauses with Skolem functions with lower arity than standard skolemization, see [Nonnengart and Weidenbach 2001] (Chapter 6 of this Handbook) for a detailed discussion).

As G. Mints pointed out, Maslov's inverse method can be seen as an attempt to search for a sequent proof from the axioms to the formula F to be proven, where labels are used instead of their corresponding nonatomic formulae (see also [Voronkov 1988]). The derivation of F in the inverse method can be easily interpreted as a resolution derivation of L_F from the set of input clauses $\delta_p(F)$.

End of Historical Remarks.

6.6. DEFINITION. Let F be a closed formula. A *definitional resolution proof* of F is a resolution derivation of L_F from $\delta(F)$.

The usual notion of a refutation can be obtained by simply adding $\neg L_F$ to $\delta(F)$. We prefer the notion of a definitional resolution proof because of its correspondence with proofs in LK.

6.7. THEOREM. *Let F be a closed formula. Then:*

$$F \text{ is valid iff } \delta_p(F) \rightarrow L_F \text{ is valid.}$$

This theorem can be proved in various ways, e.g., a proof for a slightly modified variant of $\delta_p(\cdot)$ can be found in [Plaisted and Greenbaum 1986]. Alternatively, the proof for the intuitionistic case (Theorem 4.1, Lemma 4.1 and Lemma 4.4) in [Egly 1997] can be extended to the classical case. The latter proof is based on transformations of proofs. The key steps are as follows.

- I. Transform a proof of $\vdash \delta_p(F) \rightarrow L_F$ into a proof of $\vdash F$ (see the proof sketch of Lemma 7.2 for the details for LJ).
- II. Transform a proof of $\vdash F$ into a proof of $\vdash \delta_p(F) \rightarrow L_F$.

In the following two subsections, we will compare nonstructural with the definitional transformation and the p-definitional transformation with the definitional transformation, respectively for *valid* input formulae. Our criterion for the comparison is *not* the size but the Herbrand complexity of the resulting normal form (see Definition 5.14 in Section 5). Herbrand complexity can be interpreted as a measure for the "logical hardness" of a first-order formula, i.e., how hard is it to write down a shortest (cut-free) proof.

6.1. Nonstructural versus Structural Transformations to CNFs

We first perform a comparison between a nonstructural transformation and the definitional transformation. We then sketch a similar comparison between a nonstructural transformation and the p-definitional transformation yielding essentially the same result. It will be crucial for the latter comparison that equivalences can be handled directly by the p-definitional transformation (instead of rewriting $A \leftrightarrow B$ into $A \rightarrow B \wedge B \rightarrow A$).

For our purpose, we use and extend a sequence $(F_n)_{n \in \mathbb{N}}$ of first-order formulae originally proposed by Orevkov [1979].

6.8. DEFINITION. Let F_k occur in the infinite sequence of formulae $(F_k)_{k \in \mathbb{N}}$ where

$$\begin{aligned} F_k : & (\forall x) (((\forall w_0)(\exists v_0) p(w_0, x, v_0) \wedge \\ & (\forall uvw) ((\exists y) (p(y, x, u) \wedge (\exists z) (p(v, y, z) \wedge p(z, y, w))) \rightarrow p(v, u, w))) \\ & \rightarrow (\exists v_k) (p(x, x, v_k) \wedge (\exists v_{k-1}) (p(x, v_k, v_{k-1}) \wedge \dots \wedge (\exists v_0) p(x, v_1, v_0) \dots))). \end{aligned}$$

Intuitively, $p(x, y, z)$ represents the relation $x + 2^y = z$, and F_k "computes" certain numbers using a recursive definition of this relation. Abbreviations shown in Figure 3 are used in the following in order to simplify the notation. Using these abbreviations, F_k can be rewritten to

$$(\forall x) ((A_0(x) \wedge C(x)) \rightarrow B_k(x)).$$

If $\neg F_k$ is transformed into CNF using the standard nonstructural transformation, the resulting clause set consists of three Horn clauses.

The following lemma specifies the logical complexity of some of the formulae from Figure 3, and of F_k .

6.9. LEMMA. Let $C(\alpha), B_k(\alpha), A_k(\alpha)$ be the formulae defined in Figure 3, and let F_k be the formula defined above. Then, for any $k \geq 0$, the following relations hold:

- (i) $\text{comp}(C(\alpha)) = 8$;
- (ii) $\text{comp}(B_k(\alpha)) = 2k + 1$;
- (iii) $\text{comp}(A_k(\alpha)) = 3 \cdot 2^{k+1} - 4$;
- (iv) $\text{comp}(F_k) = 2k + 14$.

PROOF. (i) follows by a straightforward calculation, and (ii)–(iv) by simple inductive arguments. \square

We start with some results about proofs of F_n in sequent calculi and resolution. The following lemma is a corollary of Theorem 1 in [Orevkov 1979].

6.10. LEMMA. Let ψ_n be a cut-free LK-proof of $\vdash F_n$. Then,

$$h(\psi_n) \geq 2 \cdot s(k) + 1 \quad \text{and} \quad l(\psi_n) \geq 2 \cdot s(k) + 1.$$

$$\begin{aligned}
 C_1(\alpha, \beta, \gamma) &: (\exists z) (p(\alpha, \beta, z) \wedge p(z, \beta, \gamma)) \\
 C_2(\alpha, \beta, \gamma, \delta) &: (\exists y) (p(y, \delta, \alpha) \wedge C_1(\beta, y, \gamma)) \\
 C(\alpha) &: (\forall u)(\forall v)(\forall w) (C_2(u, v, w, \alpha) \rightarrow p(v, u, w)) \\
 D_0(\alpha, \beta) &: (\exists v_0) p(\beta, \alpha, v_0) \\
 D_{i+1}(\alpha, \beta) &: (\exists v_{i+1}) (p(\beta, \alpha, v_{i+1}) \wedge D_i(v_{i+1}, \beta)) \\
 B_0(\alpha) &: D_0(\alpha, \alpha) \\
 B_{i+1}(\alpha) &: (\exists v_{i+1}) (p(\alpha, \alpha, v_{i+1}) \wedge D_i(v_{i+1}, \alpha)) \\
 A_0(\alpha) &: (\forall w_0)(\exists v_0) p(w_0, \alpha, v_0) \\
 A_{i+1}(\alpha) &: (\forall w_{i+1}) (A_i(w_{i+1}) \rightarrow \bar{A}_{i+1}(w_{i+1}, \alpha)) \\
 \bar{A}_0(\alpha, \beta) &: (\exists v_0) p(\alpha, \beta, v_0) \\
 \bar{A}_{i+1}(\alpha, \beta) &: (\exists v_{i+1}) (A_i(v_{i+1}) \wedge p(\alpha, \beta, v_{i+1}))
 \end{aligned}$$

Figure 3: Abbreviations used in the following.

6.11. LEMMA ([Orevkov 1979]). *There exists an LK-proof ψ_n of $\vdash F_n$ with one application of the cut rule where $l(\psi_n) \leq 41n + 7$.*

We denote the cut formula of ψ_n by $A_n(\alpha)$, where α is a free variable. The formulae $A_n(\alpha)$ do not contain equivalences.

Below we define some complexity measures for resolution derivations.

6.12. DEFINITION. Let ϕ be the resolution derivation C_1, \dots, C_n . Then we define $l(\phi) = n$ and call $l(\phi)$ the *length* of ϕ .

6.13. DEFINITION (*resolution complexity*). Let \mathcal{C} be an unsatisfiable set of clauses. Then the number

$$\mathcal{RC}(\mathcal{C}) = \min\{l(\phi) \mid \phi \text{ is a resolution refutation of } \mathcal{C}\}$$

is called the *resolution complexity* of \mathcal{C} .

Note that, by the completeness of resolution, $\mathcal{RC}(\mathcal{C})$ is always defined for unsatisfiable sets of clauses \mathcal{C} . Another measure for the "logical hardness" of sets of clauses, which is independent from the inference system, is Herbrand complexity; this concept can be defined in the same way as for sequents (see Definition 5.14).

6.14. DEFINITION (*Herbrand complexity*). Let $\mathcal{C} := \{C_1, \dots, C_n\}$ be an unsatisfiable set of clauses and let $\forall C_1, \dots, \forall C_n$ be the representations of C_1, \dots, C_n in

predicate logic. Then the *Herbrand complexity* of \mathcal{C} , denoted by $\mathcal{HC}(\mathcal{C})$, is defined to be the Herbrand complexity of the sequent

$$S(\mathcal{C}) : \forall C_1, \dots, \forall C_n \vdash .$$

Note that, for unsatisfiable sets of clauses \mathcal{C} , the sequent $S(\mathcal{C})$ is valid. Thus, by Herbrand's theorem, $\mathcal{HC}(\mathcal{C})$ is well-defined. It is easy to verify that $\mathcal{HC}(\mathcal{C})$ is the minimal number of ground instances of clauses in \mathcal{C} needed to obtain an unsatisfiable set of ground clauses. In particular, there exists always a finite unsatisfiable set of ground instances from clauses in \mathcal{C} .

6.15. DEFINITION (*Herbrand set*). Let \mathcal{C} be an unsatisfiable set of clauses and \mathcal{C}' be an unsatisfiable set of ground instances of clauses in \mathcal{C} . Then \mathcal{C}' is called a *Herbrand set* of \mathcal{C} .

There is a direct connection between Herbrand sequents and Herbrand sets: the formulae occurring in an Herbrand sequent of $S(\mathcal{C})$ just represent the ground clauses of Herbrand sets.

6.16. LEMMA ([Orevkov 1979]). *Any resolution refutation of $\gamma_{\text{NS}}(\neg F_n)$ has length greater than $s(n-1) - \log_2(n+5)$.*

The Herbrand complexity of $\gamma_{\text{NS}}(\neg F_n)$ can be estimated as follows.

$$\mathcal{HC}(\gamma_{\text{NS}}(\neg F_n)) \geq \sqrt{\mathcal{RC}(\gamma_{\text{NS}}(\neg F_n))}$$

This estimation is justified by the following three facts.

1. $\gamma_{\text{NS}}(\neg F_n)$ is a set of Horn clauses.
2. There exist resolution refutations of a ground Horn clause set of length at most quadratic in its length. In our case, the Herbrand set of $\gamma_{\text{NS}}(\neg F_n)$ is a set of ground Horn clauses.²
3. A ground resolution refutation of the Herbrand set of $\gamma_{\text{NS}}(\neg F_n)$ can be lifted to a resolution refutation of $\gamma_{\text{NS}}(\neg F_n)$. Any of these resolution refutations has length $\geq \mathcal{RC}(\gamma_{\text{NS}}(\neg F_n))$.

6.17. DEFINITION. Let $(G_n)_{n \in \mathbb{N}}$ be the infinite sequence of formulae where

$$G_n := (L \vee \neg L \vee \forall C_n) \rightarrow F_n.$$

The (universal closure of the) cut formula, $\forall C_n$, in the short LK-proof of $\vdash F_n$ is added but in such a way that the resulting clauses from this extension cannot improve Herbrand complexity. The reason is the generation of (only) tautological clauses from the lefthandside of the implication. A similar encoding of necessary information for analytic cuts is also used in [Baaz et al. 1994] (see also Section 5 for another encoding of cut).

²Recall that clauses are sets of literals.

6.18. LEMMA. *There exists an LK-proof ψ_n of $\vdash G_n$ with one application of an analytic cut such that $|\psi_n| \leq c \cdot 2^{d \cdot n}$ for some constants c and d .*

PROOF. By Lemma 6.11, there exist proofs ψ_n of $\vdash F_n$ with $l(\psi_n) \leq e \cdot n$ for some constant e . Since the size of the cut formula C_n is $\leq c_1 \cdot 2^{d_1 \cdot n}$ for some constants c_1, d_1 , there exist constants c, d such that $|\psi_n| < c \cdot 2^{d \cdot n}$. \square

Let us first estimate the Herbrand complexity of $\gamma_{\text{NS}}(\neg G_n)$.

6.19. LEMMA. *Let \mathcal{C} be an unsatisfiable set of clauses. Let \mathcal{C}^* be an unsatisfiable set of ground instances of \mathcal{C} . If there exists a set \mathcal{D} of ground clauses such that*

(i) $\mathcal{D} \subset \mathcal{C}^*$, and

(ii) $\mathcal{D} \sim_{\text{sat}} \mathcal{C}^*$

then \mathcal{C}^ cannot be a minimal Herbrand set of \mathcal{C} .*

PROOF. Obvious. \square

The lemma tells us that, for instance, tautological clauses, clauses with pure literals, and subsumed clauses cannot occur in minimal Herbrand sets because deleting such clauses preserves satisfiability. In a ground clause set, a literal is *pure*, if it occurs in one polarity only in the clause set.

6.20. LEMMA. $\mathcal{HC}(\gamma_{\text{NS}}(\neg G_n)) = \mathcal{HC}(\gamma_{\text{NS}}(\neg F_n))$.

PROOF. First of all, by Condition 4 in Definition 6.1, any clause containing literals from C_n is a tautological clause of the form $L \vee \neg L \vee D$ for a clause D . Tautological clauses, however, cannot appear in a *minimal* Herbrand set which is used to define Herbrand complexity. As a consequence, no clause from $\gamma_{\text{NS}}(\forall(L \vee \neg L \vee C_n))$ occurs in a minimal Herbrand set and $\mathcal{HC}(\gamma_{\text{NS}}(\neg G_n)) = \mathcal{HC}(\gamma_{\text{NS}}(\neg F_n))$. \square

A similar argument is used in [Baaz et al. 1994] to obtain a similar equation (but with resolution complexity instead of Herbrand complexity) for *unrestricted* resolution because tautological clauses cannot occur in shortest unrestricted resolution refutations.

As LK-proofs are trees, a comparison with resolution proofs becomes more informative for resolution tree derivations (see Definition 2.32).

6.21. DEFINITION. Let Φ be a resolution tree derivation. Then the *length* of Φ , denoted by $l(\Phi)$, is the number of nodes in Φ .

Note that any resolution tree can be written down as a sequence in a straightforward way without increasing the length. As an immediate consequence, $\mathcal{RC}(\mathcal{C}) \leq l(\Phi)$ for every tree refutation Φ of an unsatisfiable set of clauses \mathcal{C} .

The following proposition is (partially) Proposition 3.6.2 in [Eder 1992].

6.22. PROPOSITION. *Let ϕ be a proof of $\vdash F$ in LK for some closed first-order formula F , where all cut formulae occur in C_1, \dots, C_n . Then there is a derivation Φ of L_F from $\delta(F) \cup \delta(C_1) \cup \dots \cup \delta(C_n)$ in tree resolution whose size is less than $45 \cdot |\phi|^4$.*

6.23. COROLLARY. *Let ϕ be a proof of $\vdash F$ in LK for some closed first-order formula F , where all cuts are analytic. Then there is a derivation Φ of L_F from $\delta(F)$ in tree resolution whose size is less than $45 \cdot |\phi|^4$.*

PROOF. Since all cut formulae occur as subformulae in F , the result of $\delta(C_i)$ ($1 \leq i \leq n$) is already in the definitional form of F . More precisely, $\delta(C_i) \subset \delta(F)$ and, as a consequence, a proof of $\vdash F$ in LK with the restriction that all cut formulae occur as subformulae in F can be translated into a tree resolution derivation of L_F from $\delta(F)$ with only a moderate increase of proof length. \square

Observe that, in general, Corollary 6.23 does not hold for $\delta_p(\cdot)$ instead of $\delta(\cdot)$. We discuss the details below.

Let us come back to our formulae G_n . An estimation of $\mathcal{HC}(\delta(G_n) \cup \{\neg L_{G_n}\})$ remains to be performed. If we have a tree resolution refutation of a clause set S of a known length, we can estimate the Herbrand complexity with the following lemma (mainly because a tree resolution derivation possesses a ground projection [Baaz and Leitsch 1992]).

6.24. LEMMA. *Let S be a set of clauses and let Φ be a tree resolution refutation of S . Then $\mathcal{HC}(S) \leq l(\Phi)$.*

PROOF. As in [Baaz and Leitsch 1992] for linear input resolution. \square

6.25. LEMMA. *For some constants c, d , the following inequality holds:*

$$\mathcal{HC}(\delta(G_n) \cup \{\neg L_{G_n}\}) \leq c \cdot 2^{d \cdot n}.$$

PROOF. The lemma is an immediate consequence of Lemma 6.18, Corollary 6.23, and Lemma 6.24. \square

To sum up, we eventually obtain:

6.26. THEOREM. *There exists a sequence $(G_n)_{n \in \mathbb{N}}$ of valid closed first-order formulae and constants a, b, c , such that the following hold:*

1. $|\neg G_n| \leq 2^{a \cdot n}$.
2. $\mathcal{HC}(\gamma_{\text{NS}}(\neg G_n)) \geq \sqrt{s(n-1) - \log_2(n+5)}$.
3. $\mathcal{HC}(\delta(G_n) \cup \{\neg L_{G_n}\}) \leq b \cdot 2^{c \cdot n}$.

In the following, we compare nonstructural transformation with the p-definitional transformation. It is crucial for the result below that equivalences are directly translated into clauses by the p-definitional transformation, avoiding the rewriting of $A \leftrightarrow B$ into $A \rightarrow B \wedge B \rightarrow A$.

6.27. DEFINITION. Let $(H_n)_{n \in \mathbb{N}}$ be the infinite sequence of formulae where

$$H_n := (L \vee \neg L \vee (K \leftrightarrow \forall C_n)) \rightarrow F_n.$$

As mentioned above, formulae dominated by an equivalence sign are defined to occur in both polarities. Hence, any subformula occurrences of $\forall C_n$ (including this occurrence itself) occur in $\Sigma^+(H_n)$ and in $\Sigma^-(H_n)$. As a consequence, $\delta(\forall C_n) \subset \delta_p(H_n)$ and $\mathcal{HC}(\delta_p(H_n) \cup \{\neg L_{H_n}\}) \leq c \cdot 2^{d \cdot n}$ for some constants c, d . On the other hand, $\mathcal{HC}(\gamma_{\text{NS}}(\neg H_n)) = \mathcal{HC}(\gamma_{\text{NS}}(\neg G_n))$.

6.28. THEOREM. *There exists a sequence $(H_n)_{n \in \mathbb{N}}$ of valid closed first-order formulae and constants a, b, c , such that the following hold:*

1. $|\neg H_n| \leq 2^{a \cdot n}$.
2. $\mathcal{HC}(\gamma_{\text{NS}}(\neg H_n)) \geq \sqrt{s(n-1) - \log_2(n+5)}$.
3. $\mathcal{HC}(\delta_p(H_n) \cup \{\neg L_{H_n}\}) \leq b \cdot 2^{c \cdot n}$.

The reason why we need equivalences in order to get the result above is the use of subformula *occurrences* in the p-definitional transformation. In order to reduce the number of distinct labels, it is possible to optimize structural transformations. Instead of subformula occurrences, subformulae can be used with the effect that syntactically identical subformula occurrences get the same label. This optimization can even be strengthened by introducing the same label for subformula occurrences identical up to the renaming of bound variables. It is easy to establish a nonelementary "speed-up" in proof length caused by these optimizations. Two sequences of formulae with the following elements can be used.

$$\begin{aligned} G'_n &:= (L \vee \neg L \vee \forall (C_n \rightarrow C_n)) \rightarrow F_n \\ G''_n &:= (L \vee \neg L \vee \forall (C_n \rightarrow C'_n)) \rightarrow F_n \end{aligned}$$

In G''_n , C'_n is C_n modulo the name of bound variables. We will return to these optimizations of a p-definitional transformation when we discuss the effect of skolemization prior to an optimized p-definitional transformation.

6.2. Comparing Different Structural Transformations

In this subsection, we compare the p-definitional transformation with the definitional transformation. The exposition is based on results in [Egly 1996] with the difference that the fundamental complexity measure is the length instead of the size of formulae, proofs, etc. The former transformation is usually preferred in practical applications because the length of the p-definitional form of an input formula F is smaller than the length of the definitional form of the same formula. It will turn out that, from a worst case point of view, such an "optimization" is harmful mainly because a definitional transformation allows for a short simulation of analytic cuts even in cut-free calculi like LK without the cut rule. Such a short simulation is not

possible in general if a p-definitional form is preferred and the cut formula C occurs in F in one polarity p only. Since p-definitional transformation preserves polarity (e.g., either C_C^+ or C_C^- is available depending on p), only "one half" of the defining equivalences occurs in the resulting normal form. This, however, is not sufficient for a short simulation of cut because the cut formula occurs in both polarities.

6.29. DEFINITION. Let $(I_n)_{n \in \mathbb{N}}$ be the infinite sequence of formulae where

$$I_n := (M \vee \forall C_n) \rightarrow F_n$$

and M is a propositional variable not occurring in C_n and F_n .

The only difference between G_n from Definition 6.17 in Section 6.1 and I_n is the replacement of a "tautological clause" $L \vee \neg L$ (in G_n) by a pure literal M (in I_n). The following lemma for I_n can be proven similarly to Lemma 6.25 for G_n .

6.30. LEMMA. *For some constants c, d , the following inequality holds:*

$$\mathcal{HC}(\delta(I_n) \cup \{\neg L_{I_n}\}) \leq c \cdot 2^{d \cdot n}.$$

Let us estimate the Herbrand complexity of $\delta_p(I_n) \cup \{\neg L_{I_n}\}$. This is done in two steps.

6.31. LEMMA. $\mathcal{HC}(\delta_p(I_n) \cup \{\neg L_{I_n}\}) = \mathcal{HC}(\delta_p(F_n) \cup \{\neg L_{F_n}\})$.

PROOF. Note that any occurrence of a subformula is abbreviated by a unique label.

$$I_n := (M \vee \underbrace{\forall C_n}_{L_\forall}) \rightarrow F_n$$

The following clauses

$$C_1 = \neg L_\forall \vee L_\forall \vee L_M$$

$$C_2 = \neg L_M \vee M$$

occur in $\delta_p(I_n)$ and the propositional variable M occurs in C_2 only. Let $S_n := \delta_p(I_n) \cup \{\neg L_{I_n}\}$ and consider an arbitrary unsatisfiable set \mathcal{T}_n of ground instances of clauses from S_n . Note that M is pure in \mathcal{T} . As a consequence, C_2 can be removed from \mathcal{T} without affecting unsatisfiability. By this removal of C_2 , the literal L_M becomes pure in the resulting clause set and C_1 is removed. By these removals, all ground instances of such clauses introduced for subformula occurrences of $\forall C_n$ can be stepwisely deleted because they have pure literals. Moreover, clauses containing the literal L_\forall are removed. Eventually, we get \mathcal{T}' representing ground instances of $\delta_p(F_n) \cup \{\neg L_{F_n}\}$ and \mathcal{T}' is unsatisfiable. Hence, $\mathcal{HC}(\delta_p(I_n) \cup \{\neg L_{I_n}\}) = \mathcal{HC}(\delta_p(F_n) \cup \{\neg L_{F_n}\})$. \square

6.32. LEMMA.

$$\frac{\mathcal{HC}(\gamma_{\text{NS}}(\neg F_n))}{3} \leq \mathcal{HC}(\delta_p(F_n) \cup \{\neg L_{F_n}\}).$$

PROOF. Let $\mathcal{S}_n := \delta_p(F_n) \cup \{\neg L_{F_n}\}$ and let $\mathcal{T}_n := \{C_1\sigma_1, \dots, C_l\sigma_l, \neg L_{F_n}\}$ be an unsatisfiable set of ground instances of clauses occurring in \mathcal{S}_n . Let $C_i := L_i \vee C'_i$ ($i = 1, \dots, l$) with $L_i \in \{L_H(x_1, \dots, x_m), \neg L_H(x_1, \dots, x_m)\}$ for a subformula occurrence H of F_n and C_i is obtained from C_H^+ or C_H^- . Let \widehat{H} be the matrix of the prenex form of $sk_A(H)$ or the prenex form of $skr_A(H)$ depending on whether C_i is obtained from C_H^+ or C_H^- . Moreover, let \widehat{F}_n and \widehat{F}_n^r be the matrix of the prenex form of $sk_A(F_n)$ and $skr_A(\neg F_n)$, respectively. All free variables in \widehat{H} are replaced by ground terms. Replace any $C_i\sigma_i \in \mathcal{T}_n$ ($i = 1, \dots, l$) by a ground instance of $\widehat{F}_n\sigma_i\mu_i$ such that the ground subformula $\widehat{H}\sigma_i\lambda_i$ occurs in $\widehat{F}_n\sigma_i\mu_i$. The second ground substitutions λ_i and μ_i are needed to ground the remaining part of \widehat{H} and \widehat{F}_n , respectively, i.e., any remaining variable is replaced by a common constant. Therefore,

$$\widehat{F}_n\sigma_1\mu_1 \vee \dots \vee \widehat{F}_n\sigma_l\mu_l$$

is valid. Since $\neg skr_A(\neg F) = \neg sk_A(F)$ (provided the same Skolem function symbols are used),

$$\neg \widehat{F}_n^r\sigma_1\mu_1 \vee \dots \vee \neg \widehat{F}_n^r\sigma_l\mu_l$$

is valid and

$$\{\widehat{F}_n^r\sigma_1\mu_1, \dots, \widehat{F}_n^r\sigma_l\mu_l\}$$

is unsatisfiable. Recall that the cardinality of $\gamma_{\text{NS}}(\neg F_n)$ is 3. We estimate $\mathcal{HC}(\delta_p(F_n) \cup \{\neg L_{F_n}\})$ using the following two inequalities:

$$\mathcal{HC}(skr_A(\neg F_n)) < \mathcal{HC}(\delta_p(F_n) \cup \{\neg L_{F_n}\}) \quad (6.4)$$

$$\mathcal{HC}(skr_A(\neg F_n)) \cdot 3 \geq \mathcal{HC}(\gamma_{\text{NS}}(\neg F_n)) \quad (6.5)$$

From (6.4) and (6.5), the desired result follows. \square

To sum up, we eventually obtain:

6.33. THEOREM. *There exists a sequence $(I_n)_{n \in \mathbb{N}}$ of valid closed first-order formulae and constants a, b, c , such that:*

1. $|\neg I_n| \leq 2^{a \cdot n}$.
- 2.

$$\frac{\mathcal{HC}(\gamma_{\text{NS}}(\neg F_n))}{3} \leq \mathcal{HC}(\delta_p(I_n) \cup \{\neg L_{I_n}\}).$$

3. $\mathcal{HC}(\delta(I_n) \cup \{\neg L_{I_n}\}) \leq b \cdot 2^{c \cdot n}$.

There is a simpler method to show a nonelementary difference between $\delta_p(\cdot)$ and $\delta(\cdot)$ for *first-order* formulae which depends on the notion of a subformula. In the remainder of this section, we first present this method. Then we investigate the effect of skolemization prior to the application of an *optimized* p-definitional translation. A more detailed discussion can be found in [Egly 1999].

Let us reconsider F_n and the cut formula C_n . We construct a new formula in the following. First recall that the only predicate symbol occurring in F_n is p with arity 3. We replace any occurrence of an instance of $p(x, y, z)$ in F_n by a corresponding instance of $q(x, y, z, x_d)$, where x_d is a new variable. Let F'_n denote the resulting formula. Similarly, we translate the formula $C_n \rightarrow C'_n$ into $C'_n \rightarrow C'_n$ but with one important difference. Instead of x_d , we use a new variable x_c . Then J_n is defined to be

$$((\exists x_c)(\forall x)(C'_n \rightarrow C'_n)) \rightarrow (\forall x_d)F'_n.$$

Observe that J_n is closed and both indicated newly introduced quantifiers for x_c and x_d are *strong*. Clearly, if these quantifiers are eliminated in the course of a proof (say in LK without cuts) then each one is replaced by an eigenvariable. As a consequence, the left part and the right part of the topmost implication are completely separated, because $q(\cdot, \cdot, \cdot, \alpha)$ may be interpreted as $r(\cdot, \cdot, \cdot)$ and $q(\cdot, \cdot, \cdot, \beta)$ may be interpreted as $s(\cdot, \cdot, \cdot)$ for new distinct predicate symbols r and s , and eigenvariables α and β . Now, we have two possibilities to prove J_n : either to show $(\forall x_d)F'_n$ or to show the negation of $(\exists x_c)(\forall x)(C'_n \rightarrow C'_n)$. The former proof is possible but nonelementary. The question is whether we can find a shorter proof of the negation of $(\exists x_c)(\forall x)(C'_n \rightarrow C'_n)$. Since the sequent

$$\vdash (\exists x_c)(\forall x)(C'_n \rightarrow C'_n)$$

is valid (and therefore derivable in LK), the sequent

$$(\exists x_c)(\forall x)(C'_n \rightarrow C'_n) \vdash$$

cannot be derivable. Otherwise, the empty sequent \vdash is derivable (by an application of cut) which contradicts the consistency of LK (i.e., the empty sequent is not derivable).

What happens if we apply $\delta_p(\cdot)$ and $\delta(\cdot)$ to J_n ? Does it make any difference? The answer is "Yes, it makes a nonelementary difference" and we explain the answer in detail below.

Let us start with a p-definitional transformation. The following labels are introduced for the lefthandside of the topmost implication:

$$\underbrace{\underbrace{(\exists x_c)(\forall x) \left(\underbrace{C'_n}_{L_{C_n,1}(x, x_c)} \rightarrow \underbrace{C'_n}_{L_{C_n,2}(x, x_c)} \right)}_{L_{\rightarrow}(x, x_c)}}_{L_{\forall}(x_c)} \underbrace{\quad}_{L_{\exists}}$$

The following clauses are present (among others):

$$\begin{array}{ll}
 C_1 : L_{C_n,1}(x, x_c) \vee \dots & C_2 : \neg L_{C_n,2}(x, x_c) \vee \dots \\
 C_3 : \neg L_{\rightarrow}(x, x_c) \vee \neg L_{C_n,1}(x, x_c) \vee L_{C_n,2}(x, x_c) & C_4 : \neg L_{\forall}(x_c) \vee L_{\rightarrow}(x, x_c) \\
 C_5 : \neg L_{\exists} \vee L_{\forall}(c) & C_6 : L_{J_n} \vee \neg L_{\forall x_d F'_n} \\
 C_7 : L_{J_n} \vee L_{\exists}
 \end{array}$$

Recall that any subformula occurrence gets its unique label and that neither C_n nor C'_n contains equivalences. Moreover, observe that the only resolution possibility for $\neg L_{\forall}(x_c)$ in C_4 is $L_{\forall}(c)$ in C_5 . Therefore, if ground instances of C_1, \dots, C_4 occur in a Herbrand set then all variable occurrences of x_c in these instances must be instantiated with c . But then, all literal occurrences in clauses from both occurrences of C'_n have a new constant different from the constant introduced in F'_n . As a consequence, any ground instance from the p-definitional translation of $(\exists x_c)(\forall x)(C'_n \rightarrow C''_n)$ (in negative polarity) cannot occur in a minimal Herbrand set and

$$\mathcal{HC}(\delta_p(J_n) \cup \{\neg L_{J_n}\}) = \mathcal{HC}(\delta_p(F_n) \cup \{\neg L_{F_n}\}) > \mathcal{HC}(\text{skr}_A(\neg F_n)).$$

The nonelementary lower bound in n for $\mathcal{HC}(\delta_p(J_n) \cup \{\neg L_{J_n}\})$ follows.

For $\delta(J_n) \cup \{\neg J_n\}$, the situation is quite different. Since we have an encoding of both polarities of the formula C'_n with free variables x and x_c available, a polynomial simulation of the proof of F_n with cut is possible. The reason is the availability of more general formulae in $\delta(J_n)$ and in analytic cuts than in the formula in the end sequent. Recall that, even for strong quantifier occurrences Qx , $A(t)$ is a subformula of $QxA(x)$ (for an *arbitrary* term t !) although skolemization would result in $A(s)$ for an appropriate Skolem term s .

Using J_n , we investigate whether skolemization prior to an *optimized* p-definitional transformation is beneficial with respect to the minimal proof length of the resulting normal form. We introduce the same label for all syntactically identical occurrences of the same subformula. Reconsider J_n :

$$((\exists x_c)(\forall x)(C'_n \rightarrow C''_n)) \rightarrow (\forall x_d)F'_n.$$

It is important that both occurrences of C'_n (in different polarities) are abbreviated by the same label. Consequently, equivalences are introduced for any subformula of C'_n , but for the remaining subformula occurrences, implications are used. The Herbrand complexity of the resulting normal form is exponential with respect to n , because a short simulation of the cut rule by the introduced equivalences is possible.

Let us consider skolemization prior to the optimized p-definitional transformation. The skolemization of J_n , denoted by \bar{J}_n , is then as follows:

$$((\forall x)(\bar{C}'_n \rightarrow \bar{C}''_n)) \rightarrow \bar{F}'_n.$$

Observe that any cut-free LK-proof of $\vdash \bar{J}_n$ and also the Herbrand complexity of \bar{J}_n is nonelementary. Let us now consider the optimized p-definitional transformation of \bar{J}_n denoted by \bar{J}_n^* . Since the lefthandside of the topmost implication

of \bar{J}_n^* and the righthandside are completely independent (because of the different Skolem constants c and d), it suffices to use one side without a negative effect on proof length. Since the negation of the lefthandside is not provable, the righthandside must be used. A proof is possible but nonelementary, because essentially $\delta_p(F_n) \cup \{\neg L_{F_n}\}$ has to be considered. Hence, skolemization prior to an optimized p-definitional transformation can destroy the utilization of formulae for a short simulation of the cut rule with disastrous consequences for the structure and length of the resulting proof.

The above result can be used to compare the optimized variant of $\delta_p(\cdot)$ and $\delta(\cdot)$ with the structural transformation γ_{struc} from [Leitsch 1997]. The latter transformation is intended to be an optimized and simplified version of the transformation used in [Baaz et al. 1994] which in turn is a variant of $\delta(\cdot)$. The simplification is to transform formulae in *skolemized negation normal form* into a set of clauses by introducing new labels for subformulae. The important point here is that this simplification causes a nonelementary increase of proof length for some classes of formulae.

We conclude this section with a brief discussion of the δ -rule (from tableaux), which can be considered as the dynamic counterpart to skolemization (see [Zamov 1987] for a dynamic skolemization rule for clause sets obtained by variants of the (p-)definitional translation). In (analytic) free-variable tableaux and related systems (see, e.g., [Fitting 1996] for a detailed exposition on tableaux), the δ -rule is used to remove occurrences of strong quantifiers dynamically by the introduction of appropriate Skolem functions. This is in contrast to most other calculi (like resolution) which require the input formula to be skolemized in advance. Skolemization is considered to be more efficient than a dynamic δ -rule because

- (i) it causes the removal of applications of the δ -rule from the search space, and
- (ii) it is a computationally inexpensive operation.

Although applications of the δ -rule slightly enlarge the search space, dynamic skolemization can yield a drastically better behavior if variants of tableaux are used which allow for (restricted versions of) the analytic cut rule. A candidate for such a variant is the calculus KE [D'Agostino 1992, D'Agostino and Mondadori 1994]. The reason why the δ -rule is beneficial is the destruction of cut formulae by the newly introduced Skolem terms (by skolemization) as demonstrated above.

We stress that the result does *not* depend on a specific optimized δ -rule (like the δ^+ -rule [Hähnle and Schmitt 1994], the δ^{++} -rule [Beckert, Hähnle and Schmitt 1993], or the δ^* -rule [Baaz and Fermüller 1995]); even simple forms like the δ -rule suffices. The reason for the independence from the variant of the δ -rule is simple: only Skolem constants are introduced for the additional quantifier occurrences because they do not occur in the scope of essentially universal quantifiers.

6.3. Experimental Results

So far, we compared different normal form transformations with respect to the minimal proof length of the resulting normal forms. Our main interest was to show

that some techniques enable a short simulation of required instances of the cut rule in a cut-free setting whereas other techniques are too weak for such a short simulation. Due to the extreme (nonelementary) effect of cut elimination on the length of proofs in first-order logic, the minimal size of the search space decreases in the same order since this size is clearly bounded from below by the proof length. Although the results are satisfactory from a theoretical point of view, more practical issues deserve attention too.

In real situations, the preservation of the input's formula structure is the main advantage of structure-preserving transformations.³ A good example is the unsolvability of the halting problem [Burkholder 1987, Bruschi 1991, Dafa 1993, Dafa 1994, Egly and Rath 1995, Egly and Rath 1996], which is represented by a formula with many nested implications. If nonstructural transformations are applied then the structure of the formula is completely destroyed resulting in extreme difficulties to obtain a proof automatically. For a successful proof, there seems to be a "key application" of modus ponens where complex (nonatomic) subformulae are involved. In contrast to the nonstructural normal form, this application of modus ponens can be simulated by resolution within the (p-)definitional normal form. Although (p-)definitional transformations are often considered to enlarge the search space by the introduction of labels, retaining structural information often simplifies proof search.

A broader comparison between variants of (p-)definitional transformations and nonstructural transformations is performed [Egly and Rath 1996, Egly and Rath 2000] in using a preliminary (α) version of the first-order nonclausal TPTP library, see also [Nonnengart and Weidenbach 2001] (Chapter 6 of this Handbook) for a further discussion of the practical merits of (variants of) p-definitional translations. It turns out that structure-preserving transformations can compete excellently with nonstructural transformations. More precisely, applying an optimized p-definitional transformation together with some common preprocessing steps was the most successful strategy in the comparison outperforming even nonstructural transformations with the same preprocessing steps.

7. Normal Forms in Nonclassical Logics

In this section, we briefly discuss normal forms for nonclassical logics.

7.1. Intuitionistic Logic

The idea of intuitionism was invented by Brouwer in the first few years of this century. In contrast to classical logic, the *principle of excluded middle* should not

³This is the reason why literals are usually not abbreviated by labels. Moreover, this does not cause negative effects on the length of unrestricted resolution proof. For weaker calculi like analytic cut-free (clausal) tableaux, an exponential increase of proof length is possible by this "optimization" (even in the propositional case) because analytic atomic cuts can be simulated by defining labels for atoms.

G	C_G^+ and C_G^-
atomic	$C_G^+ : (\forall \vec{x}) (G \rightarrow L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (L_G(\vec{x}) \rightarrow G)$
$\neg M$	$C_G^+ : (\forall \vec{x}) ((\neg L_M(\vec{x})) \rightarrow L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (L_G(\vec{x}) \rightarrow \neg L_M(\vec{x}))$
$H \wedge I$	$C_G^+ : (\forall \vec{x}) ((L_H(\vec{y}) \wedge L_I(\vec{z})) \rightarrow L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (L_G(\vec{x}) \rightarrow L_H(\vec{y}) \wedge (\forall \vec{z}) (L_G(\vec{x}) \rightarrow L_I(\vec{z})))$
$H \vee I$	$C_G^+ : (\forall \vec{x}) (L_H(\vec{y}) \rightarrow L_G(\vec{x})) \wedge (\forall \vec{x}) (L_I(\vec{z}) \rightarrow L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (L_G(\vec{x}) \rightarrow (L_H(\vec{y}) \vee L_I(\vec{z})))$
$H \rightarrow I$	$C_G^+ : (\forall \vec{x}) ((L_H(\vec{y}) \rightarrow L_I(\vec{z})) \rightarrow L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (L_G(\vec{x}) \rightarrow (L_H(\vec{y}) \rightarrow L_I(\vec{z})))$
$(\forall x) K$	$C_G^+ : (\forall \vec{x}) (((\forall x) L_K(\vec{x}, x)) \rightarrow L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (L_G(\vec{x}) \rightarrow (\forall x) L_K(\vec{x}, x))$
$(\exists x) K$	$C_G^+ : (\forall \vec{x}) (((\exists x) L_K(\vec{x}, x)) \rightarrow L_G(\vec{x}))$ $C_G^- : (\forall \vec{x}) (L_G(\vec{x}) \rightarrow (\exists x) L_K(\vec{x}, x))$

Figure 4: The transformation rules for intuitionistic logic.

be valid. The first calculi for (fragments of) intuitionistic logic were introduced by Glivenko, Kolmogorov and Heyting [Heyting 1930a]. It was Gentzen [Gentzen 1935] who presented a Natural Deduction calculus and, even more important, an *analytic* and *cut-free* sequent calculus for first-order intuitionistic logic together with a procedure to eliminate cuts from proofs. The resulting sequent calculus is known as LJ; it is roughly described as a restriction of the sequent calculus for classical logic:

The succedent of any sequent consists of at most one formula.

Due to this restriction, intuitionistic logic is easily identified as a subsystem of classical logic. Later, calculi for first-order intuitionistic logic were introduced allowing more than one formula in the succedent (see, e.g., [Dragalin 1988, Fitting 1983] for the details and further references). In such multi-conclusion calculi, the "intuitionistic" restriction is implemented by slightly modified succedent rules for implication, negation, and the universal quantifier, whose application cause the deletion of context information in the succedent (when read from the conclusion to the premise). An example for such a succedent rule is:

$$\frac{\Gamma_1, A, \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash \Delta_1, A \rightarrow B, \Delta_2} \rightarrow: r$$

Although LJ and the multi-conclusion calculi formalize the same logic, minimal cut-free proofs can be remarkably different. For a class of propositional formulae, the

minimal proof lengths vary exponentially (see Section 3 in [Egly and Schmitt 1999] for the details).

First semantics for intuitionistic logic were introduced by Heyting [Heyting 1930a, Heyting 1930b] (see also [Troelstra 1969]) (the Brouwer-Heyting-Kolmogorov interpretation) and, later, the topological semantics studied in depth by Rasiowa and Sikorski [1963], generally called Tarskian semantics. In the fifties and sixties, two closely related semantics were developed by Beth [Beth 1956] and Kripke [Kripke 1963]. Since we have both, a cut-free analytic sequent calculus and a suitable semantics, structural transformations (similar to the classical case) into a specific normal form and semantically motivated normal forms are available, see [Ohlbach, Nonnengart, de Rijke and Gabbay 2001, Waaler 2001] (Chapters 21 and 22 of this Handbook). We briefly describe here the structural transformation approach for intuitionistic logic based on the description in [Mints 1988, Mints 1994] and show, similarly to the classical case, a nonelementary difference in proof length between a p-definitional normal form and a definitional normal form of the same intuitionistically valid formula J_n (see [Egly 1997] for details).

In the remainder of this section, we assume that axioms of LJ are atomic, i.e., they are of the form $A \vdash A$ for an atomic formula A . This is not a severe restriction in our context, because axioms of the form $F \vdash F$ (for an arbitrary formula F) can be eliminated (and replaced by atomic atoms) in a proof α with an at most exponential expense on proof length (with respect to the maximal logical complexity of an axiom in α).

Let us start with the definitions of the p-definitional and the definitional form of a first-order formula. Since these definitions are similar to the corresponding Definition 6.5 for the classical case, we do not restate them here. The only distinction between the classical and the intuitionistic case is the use of the translation schemata in Figure 4 instead of the use of the translation schemata in Figure 2. The resulting intuitionistic normal form is slightly more complicated than the corresponding classical normal form because implications cannot be replaced in general and skolemization is not possible.

7.1.1. Comparing Different Structural Translations for Intuitionistic Logic

As in the classical case, we compare the p-definitional transformation with the definitional transformation. Since we do not have a calculus-independent complexity measure like Herbrand complexity for the intuitionistic case, we compare the minimal proof length of the different normal forms in a sequent system with atomic initial sequents. More precisely, we show that there is a sequence $(J_n)_{n \in \mathbb{N}}$ of formulae for which the following hold:

1. any cut-free LJ-proof of $\vdash \delta_p(J_n) \rightarrow L_{J_n}$ is nonelementary in n , and
2. there are cut-free LJ-proofs of $\vdash \delta(J_n) \rightarrow L_{J_n}$ of length at most double-exponential in n .

Hence, we get a nonelementary decrease of proof length if we use the definitional form instead of the p-definitional form. The search space decreases in the same order if a suitable search strategy like breadth-first search or depth-first search

with iterative deepening is applied. We first recall the definition of J_n which has been already stated in Section 6.2.

7.1. DEFINITION. Let $(J_n)_{n \in \mathbb{N}}$ be the infinite sequence of formulae where

$$J_n := ((\exists x_c)(\forall x)(C'_n \rightarrow C'_n)) \rightarrow (\forall x_d)F'_n.$$

Let us first remark that *any* cut-free LJ-proof of the sequent $S_n := \vdash J_n$ has length nonelementary in n . As explained in Section 6.2, the lefthandside of the implication is worthless for a cut-free proof of S_n and the length of the proof of J_n is greater than the shortest cut-free LJ-proof of the sequent $\vdash F_n$. Any proof of the latter kind, however, is nonelementary in n .

7.2. LEMMA. *Let α be a cut-free LJ-proof of a sequent $\vdash \delta_p(F) \rightarrow L_F$. Then there exists a cut-free LJ-proof β of the sequent $\vdash F$ and the length of β is at most double-exponential in the length of α .*

PROOF SKETCH. The proof consists of three steps:

1. Construct a proof α_1 from α such that the $\rightarrow: l$ inferences introducing formulae of the form $C \rightarrow L_G$ or $L_G \rightarrow C$ are located directly below the axioms of the form $L_G \vdash L_G$.
2. Replace ("expand") every occurrence of L_G by G in α_1 and obtain a structurally equal cut-free LJ-proof α_2 with, in general, nonatomic axioms.
3. Eliminate the $\rightarrow: l$ and weakening left inferences introducing the expanded form of C_G^+ and C_G^- , respectively, and adjust the remaining proof accordingly.

The details can be found in Section 4.2 in [Egly 1997]. \square

7.3. COROLLARY. *For any cut-free LJ-proof α_n of $\vdash \delta_p(J_n) \rightarrow L_{J_n}$ and $n \geq 2$, $l(\alpha) \geq c \cdot s(n-2)$ for some constant c .*

7.4. LEMMA. *There exists a cut-free LJ-proof β_n of $\vdash \delta(J_n) \rightarrow L_{J_n}$ and $l(\beta_n) \leq c \cdot 2^{2^{d \cdot n}}$ for some constants c and d .*

PROOF. Similar to the proof of Lemma 5.4 in [Egly 1997]. \square

Corollary 7.3 and Lemma 7.4 yield the following theorem.

7.5. THEOREM. *There exists a sequence $(J_n)_{n \in \mathbb{N}}$ of intuitionistically valid closed first-order formulae and constants c and d , such that, for $n \geq 2$:*

1. *Any cut-free LJ-proof of $\vdash \delta_p(J_n) \rightarrow L_{J_n}$ has length $c \cdot s(n-2)$.*
2. *There exists a cut-free LJ-proof of $\vdash \delta(J_n) \rightarrow L_{J_n}$ with length $\leq c \cdot 2^{2^{d \cdot n}}$.*

7.2. Some Other Nonclassical Logics

In this subsection, we briefly discuss normal forms for nonclassical logics classified by the following three criteria: For the logic(s) under consideration,

1. there exist "well-behaved" (i.e., nonoperational) semantics as well as analytic (possibly cut-free) calculi;
2. there exist analytic calculi but no "well-behaved" semantics;
3. there exist "well-behaved semantics" but no analytic calculus.

In the first category, there are many well-known modal logics like S4 or K. As an example of the second and third category, we choose full linear logic and full propositional temporal logic, respectively.

7.2.1. Modal Logics

For these logics, there are quite different possibilities for structural transformations.

The most obvious ones are based on the usual transformation via the introduction of definitions (see [Mints 1988, Mints 1993a], [Voronkov 1992] for the details). There are, however, transformations of (propositional) modal logics into other logics, which can be considered as structural. Let us mention the transformation of propositional modal logic S5 into monadic (first-order) classical logic. The translation of a modal formula F into a monadic first-order formula is as follows:

1. replace all modal signs \Box , \Diamond in F by the quantifiers $\forall x$, $\exists x$, respectively, and
2. for each propositional variable p , take a unary predicate symbol r_p and replace each occurrence of p in F by $r_p(x)$.

A second structural transformation is the embedding of intuitionistic logic into the modal logic S4 [Gödel 1933, Maehara 1954, McKinsey and Tarski 1948, Rasiowa and Sikorski 1953] or [Troelstra and Schwichtenberg 1996, p. 230]. In addition to the structural normal form which exists for those modal logics with a cut-free analytic sequent calculus, there are semantic normal forms. Let us remark that structural normal forms and semantic normal forms differ in an important point: In some cases, the addition of the Barcan formula renders the resulting calculus nonanalytic, whereas constant domain simplifies semantic considerations.

7.2.2. Linear Logic

For full linear logic, there exist structural normal forms only (see, e.g., [Mints 1993b] or [Tammet 1994] for normal forms and their use in automated theorem provers) but denotational semantics do not exist.

7.2.3. Full Propositional Temporal Logics

In contrast to (full) linear logic, there exist only semantic normal forms because cut-free calculi do not exist (in general) for these logics. A possibility to explain this nonexistence is by considering complexity results for the decision problem for this temporal logic. It is shown in [Emerson 1990] that full propositional temporal logic is EXPTIME-complete. On the other hand, a usual analytic calculus would

yield a PSPACE-complete decision procedure because full contraction is available. Büchi-automata provide a form of a semantic "exposition".

8. Conclusion

It was the purpose of this article to give a proof-theoretic analysis of normal forms in automated deduction. Looking at the complexity results, we see that normal form transformations are not merely a straightforward preprocessing step, but belong to inference in an essential way. In general, the proof process is divided into the following two phases:

1. A phase which consists of the transformation of the end formula A into the corresponding normal form $N(A)$. This part is usually terminating and deterministic.
2. A phase which consists of the proof of $N(A)$.

The most promising field of future research is therefore the investigation of the connection between the benefits obtained from the clear understanding of the simplicity of phase 1. and the (possible increase of) the complexity of phase 2.

These investigations will yield a better understanding of the relation between normal forms derived from a syntactic representation and normal forms derived from a semantic representation of a given logic.

Acknowledgments

The first author is supported by the Austrian science foundation (FWF) under grant P11934-MAT.

Bibliography

- ANDREWS P. B. [1971], 'Resolution in Type Theory', *Journal of Symbolic Logic* **36**, 414–432.
- ANDREWS P. B. [1981], 'Theorem Proving via General Matings', *Journal of the ACM* **28**(2), 193–214.
- BAAZ M. AND FERMÜLLER C. G. [1995], Non-elementary Speedups between Different Versions of Tableaux, in P. Baumgartner, R. Hähnle and J. Posegga, eds, 'Proceedings of the Fourth Workshop on Theorem Proving with Analytic Tableaux and Related Methods', Springer Verlag, pp. 217–230.
- BAAZ M., FERMÜLLER C. AND LEITSCH A. [1994], A Non-Elementary Speed Up in Proof Length by Structural Clause Form Transformation, in '(Proceedings of the 9th Annual IEEE Symposium on Logics in Computer Science (LICS'94)', IEEE Computer Society Press, Los Alamitos, California, pp. 213–219.
- BAAZ M. AND LEITSCH A. [1992], 'Complexity of Resolution Proofs and Function Introduction', *Annals of Pure and Applied Logic* **57**, 181–215.
- BAAZ M. AND LEITSCH A. [1994], 'On Skolemization and Proof Complexity', *Fundamenta Informaticae* **20**, 353–379.
- BAAZ M. AND LEITSCH A. [1999], 'Cut Normal Forms and Proof Complexity', *Annals of Pure and Applied Logic* **97**, 127–177.

- BECKERT B., HÄHNLE R. AND SCHMITT P. [1993], The Even More Liberalized δ -Rule in Free Variable Semantic Tableaux, in G. Gottlob, A. Leitsch and D. Mundici, eds, 'Proceedings of the Kurt Gödel Colloquium', Vol. 713 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 108–119.
- BETH E. W. [1956], 'Semantic Construction of Intuitionistic Logic', *Kon. Nederl. Akad. Wetensch. Afd. Let. Med., Nieuwe Serie* 19(11), 357–388.
- BIBEL W. [1993], *Deduction: Automated Logic*, Academic Press, London.
- BOY DE LA TOUR T. [1990], Minimizing the Number of Clauses by Renaming, in M. E. Stickel, ed., 'Proceedings of the 10th International Conference on Automated Deduction: Kaiserslautern, 24.–27. Juli 1990', Springer Verlag, Berlin, pp. 558–572. LNAI 449.
- BOY DE LA TOUR T. [1992], 'An Optimality Result for Clause Form Translation', *Journal of Symbolic Computation* 14, 283–301.
- BRUSCHI M. [1991], 'The Halting Problem', *AAR Newsletter* pp. 7–12.
- BURKHOLDER L. [1987], 'The Halting Problem', *SIGACT News* 18(3), 48–60.
- COOK S. A. AND RECKHOW R. [1979], 'The Relative Efficiency of Propositional Proof Systems', *Journal of Symbolic Logic* 44(1), 36–50.
- COOK S. AND RECKHOW R. [1974], On the Lengths of Proofs in the Propositional Calculus, in 'Proceedings of the 5th Annual ACM Symposium on Theory of Computing', University of Toronto, pp. 135–148.
- DAFA L. [1993], 'A Mechanical Proof of the Halting Problem in Natural Deduction Style', *AAR Newsletter* pp. 4–9.
- DAFA L. [1994], 'The Formulation of the Halting Problem is Not Suitable for Describing the Halting Problem', *AAR Newsletter* pp. 1–7.
- D'AGOSTINO M. [1992], 'Are Tableaux an Improvement on Truth-Tables? Cut-Free Proofs and Bivalence', *Journal of Logic, Language and Information* 1, 235–252.
- D'AGOSTINO M. AND MONDADORI M. [1994], 'The Taming of the Cut. Classical Refutations with Analytic Cut', *Journal of Logic and Computation* 4, 285–319.
- DRAGALIN A. G. [1988], *Mathematical Intuitionism. Introduction to Proof Theory*, Vol. 67 of *Translations of Mathematical Monographs*, American Mathematical Society. Russian original 1979.
- EDER E. [1984], An Implementation of a Theorem Prover Based on the Connection Method, in W. Bibel and B. Petkoff, eds, 'AIMSA 84, Artificial Intelligence - Methodology, Systems, Applications, Varna, Bulgaria', North-Holland Publishing Company.
- EDER E. [1992], *Relative Complexities of First Order Calculi*, Vieweg, Braunschweig.
- EGLY U. [1996], 'On Different Structure-preserving Translations to Normal Form', *Journal of Symbolic Computation* 22, 121–142.
- EGLY U. [1997], 'On Definitional Translations to Normal Form for Intuitionistic Logic', *Fundamenta Informaticae* 29(1,2), 165–201.
- EGLY U. [1999], Quantifiers and the System KE: Some Surprising Results, in G. Gottlob, E. Grandjean and K. Seyr, eds, 'CSL'98', Vol. 1584 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 90–104.
- EGLY U. AND RATH T. [1995], 'The Halting Problem: An Automatically Generated Proof', *AAR Newsletter* 30, 10–16.
- EGLY U. AND RATH T. [1996], On the Practical Value of Different Definitional Translations to Normal Form, in M. McRobbie and J. K. Slaney, eds, 'Proceedings of the Conference on Automated Deduction', Springer Verlag, pp. 403–417.
- EGLY U. AND RATH T. [2000], 'On the Practical Value of Different Definitional Translations to Normal Form', *Information and Computation* 162, 255–264.
- EGLY U. AND SCHMITT S. [1999], 'On Intuitionistic Proof Transformations, Their Complexity and Application to Constructive Program Synthesis', *Fundamenta Informaticae* 39, 59–83.
- EMERSON E. A. [1990], Temporal and Modal Logic, in J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B, Elsevier, chapter 16, pp. 995–1072.

- FEFERMAN, S., DAWSON, JR., J. H., KLEENE, S. C., MOORE, G. H., SOLOVAY, R. M. AND VAN HEIJENOORT, J., EDS [1986], *Kurt Gödel Collected Works*, Vol. 1, Oxford University Press, New York.
- FERMÜLLER C. AND LEITSCH A. [1996], 'Hyperresolution and Automated Model Building', *Journal of Logic and Computation* **6**, 173–203.
- FITTING M. [1983], *Proof Methods for Modal and Intuitionistic Logics*, D. Reidel, Dordrecht.
- FITTING M. [1996], *First-Order Logic and Automated Theorem Proving*, second edn, Springer Verlag.
- GENTZEN G. [1935], 'Untersuchungen über das logische Schließen', *Mathematische Zeitschrift* **39**, 176–210, 405–431. English translation in [Szabo 1969].
- GÖDEL K. [1933], Eine Interpretation des intuitionistischen Aussagenkalküls, in K. Menger, ed., 'Ergebnisse eines Mathematischen Kolloquiums', Vol. 4, pp. 39–40. English translation in [Feferman, Dawson, Kleene, Moore, Solovay and van Heijenoort 1986], pp 300–303.
- GOUBAULT J. [1995], 'A BDD-Based Simplification and Skolemization Procedure', *Journal of the IGPL* **3**(6), 827–855.
- HÄHNLE R. AND SCHMITT P. H. [1994], 'The Liberalized δ -Rule in Free Variable Semantic Tableaux', *Journal of Automated Reasoning* **13**(2), 211–222.
- HEYTING A. [1930a], 'Die formalen Regeln der intuitionistischen Logik', *Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klassen* pp. 42–56.
- HEYTING A. [1930b], 'Die formalen Regeln der intuitionistischen Mathematik ii', *Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klassen* pp. 57–71.
- HOPCROFT J. AND ULLMAN J. [1979], *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Publishing Company, Reading, Massachusetts.
- KRIPKE S. A. [1963], 'Semantical Considerations on Modal and Intuitionistic logic', *Acta Philosophica Fennica* **16**, 83–94.
- LEITSCH A. [1997], *The Resolution Calculus*, Texts in Theoretical Computer Science, Springer Verlag.
- MAEHARA S. [1954], 'Eine Darstellung der intuitionistischen Logik in der klassischen', *Nagoya Mathematical Journal* **7**, 45–64.
- MCKINSEY J. C. AND TARSKI A. [1948], 'Some Theorems About the Sentential Calculus of Lewis and Heyting', *Journal of Symbolic Logic* **13**(1), 1–15.
- MINTS G. [1988], Gentzen-Type Systems and Resolution Rules. Part I: Propositional Logic, in P. Martin-Löf and G. Mints, eds, 'International Conference on Computer Logic', Springer Verlag, pp. 198–231.
- MINTS G. [1993a], Gentzen-Type Systems and Resolution Rules. Part II: Predicate Logic, in 'Logic Colloquium 1990', Lecture Notes in Logic, Springer Verlag.
- MINTS G. [1993b], 'Resolution Calculus for the First Order Linear Logic', *Journal of Logic, Language and Information* **2**, 59–83.
- MINTS G. [1994], Resolution Strategies for the Intuitionistic Logic, in B. Mayoh, E. Tyugu and J. Penyam, eds, 'Constraint Programming', Nato ASI Series, Springer Verlag, pp. 289–311.
- NONNENGART A., ROCK G. AND WEIDENBACH C. [1998], On Generating Small Clause Normal Forms, in 'Proceedings of the 15th International Conference on Automated Deduction (CADE-15)', Vol. 1421 of *LNAI*, Springer Verlag, pp. 397–411.
- NONNENGART A. AND WEIDENBACH C. [2001], Computing small clause normal forms, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 6, pp. 335–367.
- OHLBACH H., NONNENGART A., DE RIJKE M. AND GABBAY D. [2001], Encoding two-valued nonclassical logics in classical logic, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 21, pp. 1403–1486.

- OREVKOV V. P. [1979], 'Lower Bounds for Increasing Complexity of Derivations after Cut Elimination', *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta im V. A. Steklova AN SSSR* 88, 137–161. English translation in *Journal of Soviet Mathematics*, 2337–2350, 1982.
- PLAISTED D. A. AND GREENBAUM S. [1986], 'A Structure-Preserving Clause Form Translation', *Journal of Symbolic Computation* 2, 293–304.
- RASIOWA H. AND SIKORSKI R. [1953], 'Algebraic Treatment of the Notion of Satisfiability', *Fundamenta Mathematicae* 40, 62–95.
- RASIOWA H. AND SIKORSKI R. [1963], *The Mathematics of Metamathematics*, PWN, Warszawa.
- ROBINSON J. [1965], 'A Machine-Oriented Logic Based on the Resolution Principle', *Journal of the ACM* 12(1), 23–41.
- STATMAN R. [1979], Lower Bounds on Herbrand's Theorem, in 'Proc. AMS 75', pp. 104–107.
- SZABO, M. E., ED. [1969], *The Collected Papers of Gerhard Gentzen*, Studies in Logic and the Foundations of Mathematics, North-Holland Publishing Company.
- TAKEUTI G. [1975], *Proof Theory*, Vol. 81 of *Studies in Logic and the Foundations of Mathematics*, North-Holland Publishing Company.
- TAMMET T. [1994], 'Proof Strategies in Linear Logic', *Journal of Automated Reasoning* 12, 273–304.
- TROELSTRA A. S. [1969], *Principles of Intuitionism*, Springer Verlag.
- TROELSTRA A. S. AND SCHWICHTENBERG H. [1996], *Basic Proof Theory*, Vol. 43 of *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press.
- TSEITIN G. S. [1968], On the Complexity of Derivation in Propositional Calculus, in A. O. Slisenko, ed., 'Studies in Constructive Mathematics and Mathematical Logic, Part II', Seminars in Mathematics, V.A. Steklov Mathematical Institute, vol. 8, Leningrad, pp. 234–259. English translation: Consultants Bureau, New York, 1970, pp. 115–125.
- VORONKOV A. [1988], A Proof-search Method for the First Order Logic, in R. Martin-Löf and G. Mints, eds, 'COLOG-88', Vol. 417 of *Lecture Notes in Computer Science*, Springer, pp. 327–338.
- VORONKOV A. [1992], Theorem Proving in Non-standard Logics Based on the Inverse Method, in D. Kapur, ed., 'Proceedings of the Conference on Automated Deduction', Springer Verlag, pp. 648–662.
- WAALER A. [2001], Connections in nonclassical logics, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 22, pp. 1487–1578.
- ZAMOV N. K. [1987], 'The Resolution Method without Skolemization', *Soviet Math. Dokl.* 35(2), 399–401.

Index

A

$<_A$	297
analytic cut	282
atom	284

C

clausal form	
definitional	310
nonstructural	307
clause	284
Horn	284
clause form	
definitional	309
C_-	284
correct	
-sat	288
-val	288
strongly	288
C_+	284
cut formula	282

D

definitional form	309
intuitionistic	325
dual literal	284

E

e	283
elementary function	283
expression	284
ground	285
extension principle	310

F

formula	279
rectified	293

G

ground projection	286
-------------------------	-----

H

height	282
Herbrand complexity	303, 313
Herbrand sequent	304

I

in the scope	297
instance	285
intuitionistic logic	323, 325
irreducible	287

L

l	303
label	309
linear logic	327
literal	284
LK	280
proof	282
LK-proof	
length	303
size	303
\sim	279
logical complexity	303
logically equivalent	279

M

matrix	293
modal logic	327
more general	285

N

negation normal form	290
NNF	290
\mapsto_ν	290
nonelementary speed-up	304, 316, 319, 326
normal form	287
computation	287
operator	289

P

p-definitional form	310
\mapsto_π	293
polarity	280
predecessor	283
prenex form	293

Q

quantifier	
strong	280
weak	280

R

$\mathcal{RC}()$	313
reduction relation	287
terminating	288
resolution	285
definitional proof	311
derivation	285
refutation	285
resolution complexity	313
resolvent	285

binary285

S

s 283
 \sim_{sat} 279
 satisfiability equivalent279
 semi-terms278
 sequent280
 size303
 sk 298, 301
 Skolem form
 antiprenex298
 of a sequent299
 prenex298
 structural298, 301
 Skolemization
 of sequents302
 subformula property282
 \leq_s 285
 substitution278
 ground285
 successor283
 super proof system310

T

temporal logic327
 term278
 term depth284
 transformation
 nonstructural306
 structural306
 tree derivation286
 tree resolution286

U

unifier285
 most general285

V

\sim_{val} 279
 validity equivalent279

Computing Small Clause Normal Forms

Andreas Nonnengart

Christoph Weidenbach

SECOND READERS: Thierry Boy de la Tour and Uwe Egly.

Contents

1	Introduction	337
2	Preliminaries	338
3	Standard CNF-Translation	340
3.1	Formula Simplification	341
3.2	Negation Normal Form	341
3.3	Miniscoping	343
3.4	Variable Renaming	344
3.5	Standard Skolemization	345
3.6	Clause Normal Form	346
3.7	Clause Simplification	347
4	Formula Renaming	347
5	Skolemization	352
6	Simplification	359
6.1	Equality	360
6.2	Atom Definitions	361
6.3	Binary Decision Diagrams	362
7	Bibliographic Notes	363
8	Implementation Notes	364
	Bibliography	365
	Index	367

1. Introduction

Automated reasoning systems for first-order predicate logic usually operate on sets of clauses, see [Bachmair and Ganzinger 2001, Fermüller et al. 2001, Nieuwenhuis and Rubio 2001, Weidenbach 2001] (Chapters 2, 25, 7, and 27 of this Handbook). However, many problem formulations, in particular problems from application domains, are given in full first-order logic and thus require a transformation into clause normal form (CNF). It is well-known that the quality of such a translation has a great impact on the success of an afterwards applied automated reasoning system. From this perspective, a CNF of some formula is better than another one if it enables a system to find a proof/counter-model in a shorter period of time. Since this is an undecidable criterion, there cannot be an optimal CNF transformation algorithm in this sense. We employ the following heuristics: The smaller a set of clauses is the easier a proof or a counter-model can be found. Small can relate to different measures like the number of clauses, the number of literals, the length of the clauses or the number of different symbols. For example, in Section 4 we show that the number of generated clauses can be reduced by the introduction of new predicate symbols. There is no definite answer to the question what should be kept minimal. In a propositional context, the introduction of new propositional variables may cause the Davis-Putnam procedure to fail although it succeeds on the original formula. Even if we restrict our attention to one measure, the design of an algorithm producing “smallest” CNFs is non-trivial. For example, the introduction of new predicate symbols may cause simplification procedures to fail that would have significantly reduced the original formula. Nevertheless, experience shows that heading towards small CNFs pays off in practice. For some first-order logic formula with nested equivalences, renaming techniques (Section 4) are mandatory to effectively generate a CNF in practice. This means that there are theorem proving problems where the (doubly) exponential explosion of the number of clauses produced by a standard CNF algorithm without renaming makes the computation of the CNF intractable.

The approach we propose here emphasizes on three major aspects in the CNF transformation of first-order predicate logic formulae. These are *Formula Renaming* (Section 4), *Skolemization* (Section 5) and *Simplification* (Section 6).

By *Renaming* we mean the replacement of subformulae with some new predicates and adding a suitable definition for these new predicates. This is done whenever there is evidence that such a step will ultimately lead to fewer and smaller clauses. It shows that this method can be turned into efficient algorithms and produces smaller clause sets compared with other techniques described in the literature (see Section 7).

Moreover, we propose two advanced *Skolemization* techniques, namely *optimized* and *strong* Skolemization which turned out to be superior to the standard Skolemization approaches, i.e., they usually result in smaller and/or more general clauses. These methods can be turned into algorithms showing a reasonable behavior for all examples we tested (see Section 8).

The article is now organized as follows: after a section on preliminaries (Section 2)

where we fix notations and introduce a standard CNF transformation procedure (Section 3), Section 4 is concerned with the renaming of formulae. In Section 5 we discuss Skolemization techniques and present methods to make them tractable in practice. We finish with bibliographic notes (Section 7) and implementation notes (Section 8).

2. Preliminaries

A first-order language is constructed over a signature $\Sigma = (\mathcal{F}, \mathcal{R})$, where \mathcal{F} and \mathcal{R} are non-empty, disjoint, in general infinite sets of function and predicate symbols, respectively. Every function or predicate symbol has some fixed arity. In addition to these sets that are specific for a first-order language, we assume a further, infinite set \mathcal{X} of variable symbols disjoint from the symbols in Σ . Then the set of all *terms* $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is recursively defined by: (i) every *constant* symbol $c \in \mathcal{F}$ with arity zero is a term, (ii) every variable $x \in \mathcal{X}$ is a term and (iii) whenever t_1, \dots, t_n are terms and $f \in \mathcal{F}$ is a function symbol with arity n , then $f(t_1, \dots, t_n)$ is a term. If t_1, \dots, t_n are terms and $R \in \mathcal{R}$ is a predicate symbol with arity n , then $R(t_1, \dots, t_n)$ is an *atom*. We recursively construct *formulae* over atoms, the logical constants \top (truth), \perp (falsity) and the operators \supset (implication), \equiv (equivalence), \wedge (conjunction), \vee (disjunction), \neg (negation) and the quantifiers \forall (universal), \exists (existential) as usual: (i) every atom is a formula, (ii) \top and \perp are formulae. (iii) if ϕ_1 and ϕ_2 are formulae, so are $\phi_1 \equiv \phi_2$, $\phi_1 \supset \phi_2$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ and $\neg \phi_1$ and (iv) if ϕ is a formula and $x \in \mathcal{X}$ a variable, then $\forall x \phi$ and $\exists x \phi$ are formulae. An atom or the negation of an atom is called a *literal*. For convenience, we often write $\forall x_1, \dots, x_n \phi$ instead of $\forall x_1 \dots \forall x_n \phi$ and analogously for the existential quantifier. For the sequence $x_1, \dots, x_n (t_1, \dots, t_n)$ we use the abbreviation $\bar{x}_n (\bar{t}_n)$. Furthermore, for associative operators we often omit parentheses as in $\phi_1 \vee \phi_2 \vee \phi_3$. Disjunctions of literals are *clauses* where all variables are implicitly universally quantified. Clauses are often denoted by their respective multisets of literals where we write multisets in usual set notation.

An *interpretation* is a triple $\mathcal{M} = \langle \mathcal{D}, \mathcal{I}, \nu \rangle$ where \mathcal{D} is a non-empty set, the domain of discourse, \mathcal{I} associates n -ary predicate symbols from \mathcal{R} and function symbols from \mathcal{F} with n -place relations and functions respectively. The function $\nu : \mathcal{X} \mapsto \mathcal{D}$ is a variable valuation that associates variable symbols with concrete values taken from \mathcal{D} . By $\nu[x/a]$ we denote the variable valuation that is exactly like ν except that it maps the variable x to the domain value a . We sometimes use $\mathcal{M}[x/a]$ as a short form for $\langle \mathcal{D}, \mathcal{I}, \nu[x/a] \rangle$ provided $\mathcal{M} = \langle \mathcal{D}, \mathcal{I}, \nu \rangle$. Given an interpretation \mathcal{M} and a term t we define the interpretation of t with respect to \mathcal{M} – denoted by $\mathcal{M}(t)$ – recursively as (i) $\mathcal{M}(x) = \nu(x)$, (ii) $\mathcal{M}(f(t_1, \dots, t_n)) = \mathcal{I}(f)(\mathcal{M}(t_1), \dots, \mathcal{M}(t_n))$. An interpretation \mathcal{M} *satisfies* a formula ϕ if and only if $\mathcal{M} \models \phi$, where the relation \models is defined recursively as follows

$$\begin{aligned} \mathcal{M} &\models \top, \quad \mathcal{M} \not\models \perp \\ \mathcal{M} &\models R(t_1, \dots, t_n) \quad \text{iff} \quad (\mathcal{M}(t_1), \dots, \mathcal{M}(t_n)) \in \mathcal{I}(R) \end{aligned}$$

$$\begin{aligned}
\mathcal{M} \models \neg \phi & \quad \text{iff} \quad \mathcal{M} \not\models \phi \\
\mathcal{M} \models \phi \wedge \psi & \quad \text{iff} \quad \mathcal{M} \models \phi \text{ and } \mathcal{M} \models \psi \\
\mathcal{M} \models \forall x \phi & \quad \text{iff} \quad \mathcal{M}[x/a] \models \phi \quad \text{for every } a \in \mathcal{D} \\
\mathcal{M} \models \exists x \phi & \quad \text{iff} \quad \mathcal{M}[x/a] \models \phi \quad \text{for some } a \in \mathcal{D}
\end{aligned}$$

As usual $\phi_1 \vee \phi_2$ is interpreted as $\neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi_1 \supset \phi_2$ is interpreted as $\neg\phi_1 \vee \phi_2$ and $\phi_1 \equiv \phi_2$ is interpreted as $(\phi_1 \supset \phi_2) \wedge (\phi_2 \supset \phi_1)$.

In case there exists an interpretation \mathcal{M} that satisfies ϕ , we say that ϕ is *satisfiable* and that \mathcal{M} is a *model* of ϕ . If every interpretation satisfies ϕ then ϕ is called *valid* and we write $\models \phi$. If a formula ϕ is transformed into a formula ψ by some operation (rule), we say that such a transformation *preserves satisfiability* if ϕ is satisfiable iff ψ is. We say that such an operation *preserves equivalence* if for any interpretation \mathcal{M} we have $\mathcal{M} \models \phi$ iff $\mathcal{M} \models \psi$. Note the subtle difference between these definitions. For an operation to be satisfiability preserving only the existence of a (possibly different) model is required whereas an equivalence preserving operation guarantees satisfiability in the very same model.

A *substitution* σ is a mapping from the set of variables to the set of terms such that $x\sigma \neq x$ for only finitely many $x \in \mathcal{X}$. We define the *domain* of σ to be $\text{dom}(\sigma) = \{x \mid x\sigma \neq x\}$ and the *co-domain* of σ to be $\text{cdom}(\sigma) = \{x\sigma \mid x\sigma \neq x\}$. Hence, we can denote a substitution σ by the finite set $\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ where $x_i\sigma = t_i$ and $\text{dom}(\sigma) = \{x_1, \dots, x_n\}$. An injective substitution σ where $\text{cdom}(\sigma) \subset \mathcal{X}$ is called a *variable renaming*. The application of substitutions to terms is given by $f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$ for all $f \in \mathcal{F}$ with arity n . For the purpose of this paper, we extend substitutions to formulae as follows: $P(t_1, \dots, t_n)\sigma = P(t_1\sigma, \dots, t_n\sigma)$, $(\neg\phi)\sigma = \neg(\phi\sigma)$, $(\phi_1 \circ \phi_2)\sigma = \phi_1\sigma \circ \phi_2\sigma$ where $\circ \in \{\supset, \equiv, \wedge, \vee\}$, $(\forall x \phi)\sigma = \forall x\sigma \phi\sigma$ if $x\sigma \in \mathcal{X}$ and $\forall x \phi$ otherwise, $(\exists x \phi)\sigma = \exists x\sigma \phi\sigma$ if $x\sigma \in \mathcal{X}$ and $\exists x \phi$ otherwise. In general, the extension of substitutions to formulae is problematic, because it might accidentally turn free variables into bound variables. However, we shall only make use of this definition in very restricted contexts where the desired semantics is preserved by the application of the substitution.

A clause C is said to *subsume* a clause D , if there exists a substitution σ with $C\sigma \subseteq D$. A clause C is a *condensation* of a clause D if there exists a substitution σ where $C \subset D\sigma$, C is obtained from $D\sigma$ by the deletion of multiple occurrences of identical literals and C subsumes D . Equivalently we could say that C is a condensation of a clause D if C is a proper (multiple) factor (see [Weidenbach 2001], Chapter 27 of this Handbook) of D that subsumes D . Note that we consider clauses to be multisets of literals and not sets. Hence, the clause $\{R(x, x), R(y, y)\}$ does not subsume the clause $\{R(a, a)\}$.

A *position* is a word over the natural numbers. The set $\text{pos}(\phi)$ of positions of a given formula ϕ is defined as follows: (i) the empty word $\epsilon \in \text{pos}(\phi)$ (ii) $i.\pi \in \text{pos}(\phi)$ if $\phi = \phi_1 \circ \phi_2$ and $\pi \in \text{pos}(\phi_i)$, $i \in \{1, 2\}$ where $\circ \in \{\supset, \equiv, \wedge, \vee\}$ and (iii) $1.\pi \in \text{pos}(\phi)$ if $\phi = \neg\psi$ or $\phi = \forall x \psi$ or $\phi = \exists x \psi$ and $\pi \in \text{pos}(\psi)$. Now, if $\pi \in \text{pos}(\phi)$ we define $\phi|_{\epsilon} = \phi$ and $\phi|_{i.\tau} = \phi_i|_{\tau}$ where $\phi = \phi_1 \circ \phi_2$, $\circ \in \{\supset, \equiv, \wedge, \vee\}$ and define $\phi|_{1.\tau} = \psi|_{\tau}$ if $\phi = \neg\psi$ or $\phi = \forall x \psi$ or $\phi = \exists x \psi$. We write $\psi[\phi]_{\pi}$ for $\psi|_{\pi} = \phi$. With $\psi[\pi/\phi]$, where $\pi \in \text{pos}(\psi)$, we denote the formula obtained by replacing $\psi|_{\pi}$ with ϕ .

at position π in ψ . The *length* of a position π is defined by $|\epsilon| = 0$ and $|i.\tau| = 1 + |\tau|$. Let \leq denote the usual prefix ordering on positions: $\pi \leq \tau$ if there exists a position ι with $\pi.\iota = \tau$.

The polarity of a formula occurring at position π in a formula ψ is denoted by $pol(\psi, \pi)$ and is defined in the usual way: $pol(\psi, \epsilon) = 1$; $pol(\psi, \pi.i) = pol(\psi, \pi)$ if $\psi|_\pi$ is a conjunction, disjunction, formula starting with a quantifier or an implication with $i = 2$; $pol(\psi, \pi.i) = -pol(\psi, \pi)$ if $\psi|_\pi$ is a formula starting with a negation symbol or an implication with $i = 1$ and, finally, $pol(\psi, \pi.i) = 0$ if $\psi|_\pi$ is an equivalence.

The *depth* of a formula ϕ (term t) is given by $depth(\phi) = \max(\{|\pi| \mid \pi \in pos(\phi)\})$ ($depth(t) = \max(\{|\pi| \mid \pi \in pos(t)\})$).

The set of *free* variables of a formula ϕ (term t), denoted by $free(\phi)$ is defined as follows: $free(P(t_1, \dots, t_n)) = \cup_i free(t_i)$, $free(\neg\phi) = free(\phi)$, $free(\phi_1 \circ \phi_2) = free(\phi_1) \cup free(\phi_2)$ where $\circ \in \{\supset, \equiv, \wedge, \vee\}$, $free(\forall x \phi) = free(\phi) \setminus \{x\}$, $free(\exists x \phi) = free(\phi) \setminus \{x\}$ ($free(f(t_1, \dots, t_n)) = \cup_i free(t_i)$, $free(x) = \{x\}$). A formula ϕ is a *sentence* if ϕ does not contain any free variables ($free(\phi) = \emptyset$). The set of *bound* variables of a formula ϕ (term t), denoted by $bound(\phi)$ is defined as follows: $bound(P(t_1, \dots, t_n)) = \emptyset$, $bound(\neg\phi) = bound(\phi)$, $bound(\phi_1 \circ \phi_2) = bound(\phi_1) \cup bound(\phi_2)$ where $\circ \in \{\supset, \equiv, \wedge, \vee\}$, $bound(\forall x \phi) = bound(\phi) \cup \{x\}$, $bound(\exists x \phi) = bound(\phi) \cup \{x\}$ ($bound(t) = \emptyset$). Finally, the set of all variables of a formula ϕ (term t) is defined by $vars(\phi) = bound(\phi) \cup free(\phi)$ ($vars(t) = free(t)$).

3. Standard CNF-Translation

In this section we introduce a standard way of translating first-order formulae into clause normal form. Standard means that algorithms similar to the one suggested here can be found in many textbooks on automated theorem proving or first-order logic. There are many useful techniques beyond the material presented in this chapter. In Section 4, Section 5 and Section 6 we suggest some important extensions. For further material consider the references presented in Section 7.

A sentence ϕ is in *clause normal form* (CNF), if $\phi = \forall x_1 \dots \forall x_k [C_1 \wedge \dots \wedge C_n]$ where $C_i = L_{i,1} \vee \dots \vee L_{i,l_i}$ and each $L_{i,j}$ is a literal. An alternative representation of a CNF is a set of clauses, where variables in the clauses are implicitly universally quantified, their literals disjunctively connected and all clauses form a conjunction. Clause normal form translation converts an arbitrary formula into a CNF, preserving satisfiability. Thus, important properties of the CNF-translation algorithm are termination, that it preserves satisfiability and that it generates CNFs.

Main parts of the algorithm are presented by collections of rules. The rules are either of the form $\phi_1 \rightarrow \phi_2$ or $\phi[\phi_1]_\pi \rightarrow \phi[\pi/\phi_2]$, where the former is an abbreviation for the latter, in case we are not interested in the particular position. Informally, these rules allow us to replace the subformula ϕ_1 (occurring at position π) with the formula ϕ_2 within some arbitrary formula ϕ . Collections of rules are applied in an exhaustive, don't-care non-deterministic way. Furthermore, formula matching is performed modulo the commutativity of \wedge , \vee and \equiv .

$\phi \wedge \phi \rightarrow \phi$	$\phi \vee \phi \rightarrow \phi$
$\phi \wedge \neg \phi \rightarrow \perp$	$\phi \vee \neg \phi \rightarrow \top$
$\phi \equiv \phi \rightarrow \top$	$\phi \supset \phi \rightarrow \top$
$\neg \top \rightarrow \perp$	$\neg \perp \rightarrow \top$
$\phi \wedge \top \rightarrow \phi$	$\phi \vee \top \rightarrow \top$
$\phi \wedge \perp \rightarrow \perp$	$\phi \vee \perp \rightarrow \phi$
$\phi \supset \top \rightarrow \top$	$\top \supset \phi \rightarrow \phi$
$\phi \supset \perp \rightarrow \neg \phi$	$\perp \supset \phi \rightarrow \top$
$\phi \equiv \top \rightarrow \phi$	$\phi \equiv \perp \rightarrow \neg \phi$
$\forall x \phi \rightarrow \phi \quad \text{if } x \notin \text{free}(\phi)$	
$\exists x \phi \rightarrow \phi \quad \text{if } x \notin \text{free}(\phi)$	

Table 1: Simplification Rules

The standard CNF-translation algorithm is divided into seven subsequent steps represented by the Subsections 3.1–3.7 below.

3.1. Formula Simplification

The collection in Table 1 removes the logical constants \top , \perp from positions below ϵ . Also it removes redundant quantifiers and performs some first-order simplifications like the application of idempotence of \wedge , \vee or the removal of certain syntactic tautologies.

The application of the rules terminates, since every right hand side of a rule contains fewer symbols than its left hand side. All these rules are equivalence preserving. After an exhaustive application of the collection to a formula resulting in ϕ , either $\phi \in \{\top, \perp\}$ or ϕ does not contain \top nor \perp and every quantifier in ϕ binds a variable that occurs freely in its argument formula.

If a formula is reduced to \top or \perp by the above rules, the CNF translation stops here and we are already done. Hence, for further considerations we assume that formulae contain neither \top nor \perp .

3.2. Negation Normal Form

A formula is in *negation normal form* (NNF), if it does not contain implication or equivalence symbols, and every negation symbol occurs directly in front of an atom. A transformation into NNF can be performed with the help of the rules shown in Table 2

$\psi[\phi_1 \equiv \phi_2]_\pi \rightarrow \psi[\pi/(\phi_1 \supset \phi_2) \wedge (\phi_2 \supset \phi_1)]$	if $\text{pol}(\psi, \pi) = 1$
$\psi[\phi_1 \equiv \phi_2]_\pi \rightarrow \psi[\pi/(\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \neg\phi_2)]$	if $\text{pol}(\psi, \pi) = -1$
$\neg(\phi \wedge \psi) \rightarrow \neg\phi \vee \neg\psi$	$\neg(\phi \vee \phi) \rightarrow \neg\phi \wedge \neg\psi$
$\neg(\forall x \phi) \rightarrow \exists x \neg\phi$	$\neg(\exists x \phi) \rightarrow \forall x \neg\phi$
$\phi \supset \psi \rightarrow \neg\phi \vee \psi$	$\neg\neg\phi \rightarrow \phi$

Table 2: Transformation into NNF

The rules terminate, since they either decrease the number of equivalence or implication symbols or they decrease the depth of formulae that start with a negation symbol. All these rules are equivalence preserving. They transform any formula ϕ into a NNF equivalent to ϕ .

Note that for the elimination of equivalences, we need not consider positions with polarity 0, since we can always apply these rules at a position with minimal length. The formulae at such positions always have polarity 1 or -1. The suggested transformation is called *polarity dependent* elimination of equivalence symbols. This elimination avoids generating redundant clauses that are hardly recognizable once the CNF is built: Assume that we always apply the first variant $\phi_1 \equiv \phi_2 \rightarrow (\phi_1 \supset \phi_2) \wedge (\phi_2 \supset \phi_1)$ to replace equivalence symbols, independently from the polarity of the formula. Then consider the following transformation of $\neg(\phi_1 \equiv \phi_2)$

$$\begin{aligned}
& \neg(\phi_1 \equiv \phi_2) \\
& \downarrow \\
& \neg((\phi_1 \supset \phi_2) \wedge (\phi_2 \supset \phi_1)) \\
& \downarrow \\
& \neg((\neg\phi_1 \vee \phi_2) \wedge (\neg\phi_2 \vee \phi_1)) \\
& \downarrow \\
& \neg(\neg\phi_1 \vee \phi_2) \vee \neg(\neg\phi_2 \vee \phi_1) \\
& \downarrow \\
& (\phi_1 \wedge \neg\phi_2) \vee (\phi_2 \wedge \neg\phi_1) \\
& \downarrow \\
& (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \neg\phi_1) \wedge (\neg\phi_2 \vee \phi_2) \wedge (\neg\phi_2 \vee \neg\phi_1)
\end{aligned}$$

where we used the rules for NNF and finally the distributive law to obtain the correct shape of a CNF (see also Section 3.6). Now the disjuncts $(\phi_1 \vee \neg\phi_1)$ and $(\neg\phi_2 \vee \phi_2)$ are tautologies (see Section 3.1) and can therefore be removed. However, if ϕ_1 and ϕ_2 are complex formulae this may not be easily detectable, e.g., $\neg\phi_1$ is transformed by the above rules into a formula that is no longer syntactically the negation of ϕ_1 . This problem is completely avoided by the polarity dependent elimination suggested above, since it will not generate any of these tautologies. We

shall make use of this particular transformation in our section on formula renaming (Section 4).

3.3. Miniscoping

In Section 3.5 below, we shall get rid of existential quantifiers via the introduction of Skolem functions. The arity of a Skolem function is determined by the number of universally quantified variables that appear in the argument formula of the existential quantifier to be removed. The aim of the rules here is to minimize the arity of Skolem functions by moving quantifiers as far inwards as possible. So this step is only for Skolem function argument number optimization purposes.

$\exists x (\phi \wedge \psi)$	\rightarrow	$\exists x \phi \wedge \psi$	if $x \notin \text{free}(\psi)$
$\exists x (\phi \vee \psi)$	\rightarrow	$\exists x \phi \vee \psi$	if $x \notin \text{free}(\psi)$
$\forall x (\phi \wedge \psi)$	\rightarrow	$\forall x \phi \wedge \psi$	if $x \notin \text{free}(\psi)$
$\forall x (\phi \vee \psi)$	\rightarrow	$\forall x \phi \vee \psi$	if $x \notin \text{free}(\psi)$
$\forall x (\phi \wedge \psi)$	\rightarrow	$\forall x \phi \wedge \forall x \psi$	if $x \in \text{free}(\phi)$ and $x \in \text{free}(\psi)$
$\exists x (\phi \vee \psi)$	\rightarrow	$\exists x \phi \vee \exists x \psi$	if $x \in \text{free}(\phi)$ and $x \in \text{free}(\psi)$

All rules are equivalence preserving. They terminate, since they always reduce the depth of a formula starting with a quantifier.

Here is an example that demonstrates the potential of this transformation:

$$\begin{aligned}
 & \forall x \exists y \forall z (R(x, x) \wedge (P(y) \vee R(x, y) \vee Q(z))) \\
 & \quad \downarrow \\
 & \forall x \exists y (R(x, x) \wedge \forall z (P(y) \vee R(x, y) \vee Q(z))) \\
 & \quad \downarrow \\
 & \forall x \exists y (R(x, x) \wedge (P(y) \vee R(x, y) \vee \forall z Q(z))) \\
 & \quad \downarrow \\
 & \forall x (R(x, x) \wedge \exists y (P(y) \vee R(x, y) \vee \forall z Q(z))) \\
 & \quad \downarrow \\
 & \forall x (R(x, x) \wedge (\exists y P(y) \vee \exists y R(x, y) \vee \forall z Q(z))) \\
 & \quad \downarrow \\
 & \forall x R(x, x) \wedge \forall x (\exists y P(y) \vee \exists y R(x, y) \vee \forall z Q(z)) \\
 & \quad \downarrow \\
 & \forall x R(x, x) \wedge (\exists y P(y) \vee \forall x \exists y R(x, y) \vee \forall z Q(z))
 \end{aligned}$$

In Section 3.5 we continue the example showing that this transformation enables the Skolemization rule to generate Skolem functions with fewer arguments.

3.4. Variable Renaming

Miniscoping duplicates quantifiers. Since we want to eventually move all remaining universal quantifiers completely outwards, we have to rename variables such that different occurrences of quantifiers bind different variables. Moreover, our Skolemization rule does not work properly if different quantifiers bind the same variable. Variable renaming is well-known to be an error-prone subject. Therefore, we provide a formal definition and a formal proof for the properties enjoyed by the variable renaming procedure.

Let ψ be a formula and $\bar{y} = y_1, y_2, \dots$ be an infinite sequence of different variables that do not occur in ψ . Then we recursively define the variable renaming function $ren(\psi) = ren(\psi, \bar{y}, 1)$ by

$$ren(\psi, \bar{y}, n) = \begin{cases} P(t_1, \dots, t_n) & \text{if } \psi = P(t_1, \dots, t_n) \\ ren(\phi_1, \bar{y}, n) \circ ren(\phi_2, \bar{y}, n + nq(\phi_1)) & \text{if } \psi = \phi_1 \circ \phi_2 \\ \neg ren(\phi, \bar{y}, n) & \text{if } \psi = \neg \phi \\ \exists y_n ren(\phi\{x \mapsto y_n\}, \bar{y}, n + 1) & \text{if } \psi = \exists x \phi \\ \forall y_n ren(\phi\{x \mapsto y_n\}, \bar{y}, n + 1) & \text{if } \psi = \forall x \phi \end{cases}$$

where for the second case $\circ \in \{\supset, \equiv, \wedge, \vee\}$. The function nq counts the number of quantifiers in a formula: $nq(\psi) = |\{\pi \mid \psi|_\pi = \exists x \phi \text{ or } \psi|_\pi = \forall x \phi\}|$.

The renaming of variables is a subtle issue. Therefore, we now prove in detail that the variable renaming function ren generates the desired result. In order to prove the logical equivalence between a formula and a renamed variant, we make use of the lemma below.

3.1. LEMMA. *Let σ be a variable renaming and let ϕ be a formula with $cdom(\sigma) \cap vars(\phi) = \emptyset$. Moreover, let ν_1 and ν_2 be two variable valuations satisfying $\nu_1(x) = \nu_2(x\sigma)$ for all $x \in free(\phi)$. Then*

$$\langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle \models \phi \text{ iff } \langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle \models \phi\sigma$$

PROOF. By induction on the structure of formulae. Note that if $cdom(\sigma) \cap vars(\phi) = \emptyset$ then $cdom(\sigma) \cap vars(\psi) = \emptyset$ and $cdom(\sigma) \cap vars(t) = \emptyset$ for any subformula ψ of ϕ and any term t occurring in ϕ . For the purpose of this proof we say that two valuations ν_1 and ν_2 agree modulo a variable renaming σ , if $\nu_1(x) = \nu_2(x\sigma)$ for all x . First, we prove by structural induction that $\langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle(t) = \langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle(t\sigma)$ for all terms t . If t is a variable y , then $\langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle(y) = \nu_1(y) = \nu_2(y\sigma) = \langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle(y\sigma)$ since y occurs free in y . If t is a constant c , then $\langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle(c) = \mathcal{I}(c) = \langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle(c\sigma)$. Finally, if t is a compound term $f(t_1, \dots, t_n)$ then since ν_1 and ν_2 agree on the free variables of t modulo σ , the two valuations agree on all variables of the t_i modulo σ . Therefore, we conclude $\langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle(f(t_1, \dots, t_n)) = \mathcal{I}(f)(\langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle(t_1), \dots, \langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle(t_n)) = \mathcal{I}(f)(\langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle(t_1\sigma), \dots, \langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle(t_n\sigma)) = \langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle(f(t_1, \dots, t_n)\sigma)$ by definition and by applying the induction hypothesis to the t_i .

Second, we prove the main conjecture by structural induction on formulae. For an atom $P(t_1, \dots, t_n)$, if ν_1 and ν_2 agree on the free variables of $P(t_1, \dots, t_n)$ modulo σ , they agree on all free variables of the t_i modulo σ . We

conclude $\langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle \models P(t_1, \dots, t_n)$ iff $\mathcal{I}(P)(\langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle(t_1), \dots, \langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle(t_n))$ iff $\mathcal{I}(P)(\langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle(t_1\sigma), \dots, \langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle(t_n\sigma))$ iff $\langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle \models P(t_1, \dots, t_n)\sigma$ by definition and by induction hypothesis applied to the t_i . The cases for all first-order operators except for the quantifiers are similar and are therefore omitted. For the quantifiers we concentrate on the existential quantifier. The treatment of the universal quantifier works analogous and is also omitted. We have to show that $\langle \mathcal{D}, \mathcal{I}, \nu_1 \rangle \models \exists x \phi$ iff $\langle \mathcal{D}, \mathcal{I}, \nu_2 \rangle \models (\exists x \phi)\sigma$. If $x \notin \text{dom}(\sigma)$, then $(\exists x \phi)\sigma = \exists x \phi\sigma$. For any $a \in \mathcal{D}$ the valuations $\nu_1[x/a]$ and $\nu_2[x/a]$ agree on all free variables of ϕ since they agree on all free variables of $\exists x \phi$ and obviously they agree on x . Hence, for any $a \in \mathcal{D}$ we have $\langle \mathcal{D}, \mathcal{I}, \nu_1[x/a] \rangle \models \phi$ iff $\langle \mathcal{D}, \mathcal{I}, \nu_2[x/a] \rangle \models \phi\sigma$ by induction hypothesis, implying the conjecture for the case $x \notin \text{dom}(\sigma)$. If $x \in \text{dom}(\sigma)$, where $x\sigma = y$, then by definition $(\exists x \phi)\sigma = \exists y \phi\sigma$. If ν_1, ν_2 agree on the free variables of $\exists x \phi$, the valuations $\nu_1[x/a], \nu_2[y/a]$ agree on the free variables of ϕ modulo σ for any $a \in \mathcal{D}$ because $y \notin \text{vars}(\exists x \phi)$ and σ is injective. Therefore, by the induction hypothesis, we get $\langle \mathcal{D}, \mathcal{I}, \nu_1[x/a] \rangle \models \phi$ iff $\langle \mathcal{D}, \mathcal{I}, \nu_2[y/a] \rangle \models \phi\sigma$. This implies the main conjecture and we are done. \square

3.2. LEMMA. *The function ren terminates, there are no two quantifiers in $\text{ren}(\Psi)$ binding the same variable and the formulae $\text{ren}(\Psi)$ and Ψ are logically equivalent.*

PROOF. First, all recursive calls of ren apply to proper subformulae. This guarantees termination. Second, the function renames exactly the variables bound by quantifiers. The cases four and five of the definition introduce a fresh variable y_n that will not be used in the recursive call, since n is incremented. Using an induction argument on the definition of ren , this property is preserved. In particular, for the second case where ren distributes over a first-order operator, in the recursive call for ϕ_2 the parameter n is incremented by the number of quantifier occurrences in ϕ_1 ensuring that different y_i are introduced for the two subformulae. Third, logical equivalence follows by an inductive argument on the definition of ren using Lemma 3.1. \square

Renaming the variables of

$$\forall x R(x, x) \wedge (\exists y P(y) \vee \forall x \exists y R(x, y) \vee \forall z Q(z))$$

(see Section 3.3) with respect to the sequence y_1, y_2, \dots results in

$$\forall y_1 R(y_1, y_1) \wedge (\exists y_2 P(y_2) \vee \forall y_3 \exists y_4 R(y_3, y_4) \vee \forall y_5 Q(y_5))$$

3.5. Standard Skolemization

In this step we get rid of the existential quantifiers. Note that the rule below eliminates existential quantifiers in an outermost way, since the position of the subformula under consideration is always required to be of minimal length. In addition, the function f is assumed to be new with respect to the symbols occurring in the overall formula ϕ .

$$\begin{array}{l} \phi[\exists x \psi]_{\pi} \rightarrow \phi[\pi/\psi\{x \mapsto f(y_1, \dots, y_n)\}] \\ \text{if } \text{free}(\exists x \psi) = \{y_1, \dots, y_n\}, f \text{ new, } |\pi| \text{ minimal} \end{array}$$

Skolemization terminates because every rule application removes one of the existential quantifiers. The result of a rule application is not logically equivalent to the formula the rule is applied to, but preserves satisfiability. In Section 5 we will prove this property for two advanced techniques that both subsume the above Skolemization rule.

Applying the Skolemization rule twice to our example results in the formula

$$\forall y_1 R(y_1, y_1) \wedge (P(a) \vee \forall y_3 R(y_3, f(y_3)) \vee \forall y_5 Q(y_5))$$

where a is a Skolem constant and f a Skolem function. Note that without miniscoping (Section 3.3) we would have obtained $P(f(y_1))$ instead of $P(a)$. This example nicely shows that it is a priori not known whether miniscoping is a useful operation in the sense that it supports the search of an afterwards applied proof search procedure, like resolution. If only a is used in a proof, then the miniscoping might have reduced proof complexity, given the Herbrand complexity measures introduced in [Baaz et al. 2001] (Chapter 5 of this Handbook), since we could replace a one place function by a constant. If both a and f are needed for a proof, then miniscoping might have increased proof complexity, since it caused the generation of two Skolem functions where without miniscoping only one is generated.

3.6. Clause Normal Form

Finally, we move all universal quantifiers outwards and compute a conjunctive normal form.

$$\begin{array}{lll} \forall x \phi \wedge \psi & \rightarrow & \forall x (\phi \wedge \psi) & \text{if } x \notin \text{free}(\psi) \\ \forall x \phi \vee \psi & \rightarrow & \forall x (\phi \vee \psi) & \text{if } x \notin \text{free}(\psi) \\ \phi \vee (\psi_1 \wedge \psi_2) & \rightarrow & (\phi \vee \psi_1) \wedge (\phi \vee \psi_2) \end{array}$$

All three rules are equivalence preserving and terminating since they either decrease the length of subformulae positions starting with a quantifier or the number of conjunctions that occur below the position of a disjunction.

For our running example this means

$$\forall y_1 R(y_1, y_1) \wedge (P(a) \vee \forall y_3 R(y_3, f(y_3)) \vee \forall y_5 Q(y_5))$$

$$\downarrow$$

$$\forall y_1, y_3, y_5 (R(y_1, y_1) \wedge (P(a) \vee R(y_3, f(y_3)) \vee Q(y_5)))$$

$$\downarrow$$

$$\forall y_1, y_3, y_5 ((R(y_1, y_1) \vee P(a)) \wedge (R(y_1, y_1) \vee R(y_3, f(y_3))) \wedge (R(y_1, y_1) \vee Q(y_5)))$$

where we left out some intermediate steps.

3.7. Clause Simplification

Finally, we simply remove the universal quantifiers and turn our conjunction of disjunctions of literals into a clause set. We always assume that all variables are universally quantified and that different clauses do not share any variables.

The final formula of the previous step yields the clause set

$$\{\{R(y_1, y_1), P(a)\}, \{R(y_6, y_6), R(y_3, f(y_3))\}, \{R(y_7, y_7), Q(y_5)\}\}$$

where we renamed occurrences of y_1 with y_6 and y_7 , respectively.

There is a bunch of well-known simplifications on the clause level, see [Bachmair and Ganzinger 2001, Nieuwenhuis and Rubio 2001, Weidenbach 2001] (Chapters 2, 7 and 27 of this volume). Standard techniques are the removal of subsumed clauses and tautologies. A clause is called a (syntactic) *tautology* if it contains both some atom and its negation. Subsumption as well as tautology deletion are equivalence preserving and can only be applied finitely often to a finite clause set.

4. Formula Renaming

Let us illustrate the problem with the help of a simple example. Consider the formula

$$\phi_1 \vee \forall x \phi_2$$

where we assume that x is the only free variable in ϕ_2 . If n is the number of clauses generated by ϕ_1 and m is the number of clauses generated by $\forall x \phi_2$ then the above formula generates nm clauses. Thus, a nesting of such disjunctions easily leads to an exponential increase in the number of clauses. The same holds for nested implications and equivalences. The reason for this exponential explosion is the (exponential) duplication of subformulae obtained by the exhaustive application of the distributivity law. A solution to this problem is *Formula Renaming*, i.e., the replacement of subformulae using new predicates. For the above formula, a renaming of ϕ_2 is

$$[\phi_1 \vee \forall x P(x)] \wedge \forall x (P(x) \supset \phi_2)$$

where P is a new one-place predicate. The renamed formula is not logically equivalent to the first formula, but preserves satisfiability and this suffices for resolution-based theorem proving. Furthermore, the renamed formula generates only $n + m$ clauses. Thus, using a renaming technique, the worst case exponential explosion of the number of clauses generated by a formula can be avoided.

4.1. DEFINITION (Formula Renaming). Let ψ be a formula and let $\phi = \psi|_\pi$ be the subformula of ψ we want to rename. Let x_1, \dots, x_n be the free variables in ϕ and let R be a predicate symbol with arity n which is new to ψ . Then the formula

$$\psi[\pi/R(x_1, \dots, x_n)] \wedge \text{def}(\pi, \psi, R)$$

is a *formula renaming* of ψ at position π . The formula $\text{def}(\pi, \psi, R)$ is a polarity dependent definition of the new predicate R :

$$\text{def}(\pi, \psi, R) = \begin{cases} \forall x_1, \dots, x_n [R(x_1, \dots, x_n) \supset \phi] & \text{if } \text{pol}(\psi, \pi) = 1 \\ \forall x_1, \dots, x_n [\phi \supset R(x_1, \dots, x_n)] & \text{if } \text{pol}(\psi, \pi) = -1 \\ \forall x_1, \dots, x_n [R(x_1, \dots, x_n) \equiv \phi] & \text{if } \text{pol}(\psi, \pi) = 0 \end{cases}$$

Formula renaming is a satisfiability preserving operation. This can be proved by choosing appropriate interpretations for the introduced predicates.

Now recall, that we only want to rename a formula if the number of clauses generated from the renamed formula is smaller than the number of clauses generated from the original formula. The first step towards this aim is to calculate the number of clauses generated by a standard CNF without any reduction. This is done by the function p defined in Table 3, where $\bar{p}(\psi)$ stands for $p(\neg\psi)$ for any formula ψ . The first column shows the form of ψ and the next two columns the corresponding recursive, top-down calculations for $p(\psi)$ and $\bar{p}(\psi)$, respectively.

ψ	$p(\psi)$	$\bar{p}(\psi)$
$\phi_1 \wedge \phi_2$	$p(\phi_1) + p(\phi_2)$	$\bar{p}(\phi_1)\bar{p}(\phi_2)$
$\phi_1 \vee \phi_2$	$p(\phi_1)p(\phi_2)$	$\bar{p}(\phi_1) + \bar{p}(\phi_2)$
$\phi_1 \supset \phi_2$	$\bar{p}(\phi_1)p(\phi_2)$	$p(\phi_1) + \bar{p}(\phi_2)$
$\phi_1 \equiv \phi_2$	$p(\phi_1)\bar{p}(\phi_2) + \bar{p}(\phi_1)p(\phi_2)$	$p(\phi_1)p(\phi_2) + \bar{p}(\phi_1)\bar{p}(\phi_2)$
$\forall x \phi_1$	$p(\phi_1)$	$\bar{p}(\phi_1)$
$\exists x \phi_1$	$p(\phi_1)$	$\bar{p}(\phi_1)$
$\neg \phi_1$	$\bar{p}(\phi_1)$	$p(\phi_1)$
$P(t_1, \dots, t_n)$	1	1

Table 3: Calculating the number of clauses

Note that the calculation for equivalences assumes a polarity dependent elimination of equivalences. Recall that this elimination avoids generating redundant clauses that are hardly recognizable once the CNF is built (see page 342).

Second, in order to check whether a renaming yields fewer clauses, we need to calculate the difference between the number of clauses generated with and without a renaming. So let us assume that we want to rename a subformula at position π within a formula ψ . The condition to be checked is

$$p(\psi) \geq p(\psi[\pi/R(x_1, \dots, x_n)]) + p(\text{def}(\pi, \psi, R))$$

The obvious problem with this condition is that the function p cannot be efficiently computed in general, for it grows exponentially in the size of the input formula. Moreover, a straightforward top-down implementation of p following Table 3 results

in an algorithm with exponential time complexity. The exponential complexity can be avoided using a dynamic programming idea: we simply store intermediate results for subformulae. Nevertheless, because p grows exponentially, computing p requires arbitrary precision arithmetic. It turns out that this can hardly be afforded in practice. The rest of this section is therefore concerned with a solution to this problem, i.e., we shall show that it is not necessary to compute p at all.

Obviously, the formulae ψ and $\psi[\pi/R(x_1, \dots, x_n)]$ differ only at position π , the other parts of the formulae remain identical. We make use of this fact by an abstraction of those parts of ψ that do not influence the changed position. To this end we introduce the notion of a coefficient as shown in Table 4. The coefficients

π	$\psi _\pi$	a_π^ψ	b_π^ψ
$\tau.i$	$\phi_1 \wedge \phi_2$	a_τ^ψ	$b_\tau^\psi \prod_{j \neq i} \bar{p}(\phi_j)$
$\tau.i$	$\phi_1 \vee \phi_2$	$a_\tau^\psi \prod_{j \neq i} p(\phi_j)$	b_τ^ψ
$\tau.1$	$\phi_1 \supset \phi_2$	b_τ^ψ	$a_\tau^\psi p(\phi_2)$
$\tau.2$	$\phi_1 \supset \phi_2$	$a_\tau^\psi \bar{p}(\phi_1)$	b_τ^ψ
$\tau.1$	$\phi_1 \equiv \phi_2$	$a_\tau^\psi \bar{p}(\phi_2) + b_\tau^\psi p(\phi_2)$	$a_\tau^\psi p(\phi_2) + b_\tau^\psi \bar{p}(\phi_2)$
$\tau.2$	$\phi_1 \equiv \phi_2$	$a_\tau^\psi \bar{p}(\phi_1) + b_\tau^\psi p(\phi_1)$	$a_\tau^\psi p(\phi_1) + b_\tau^\psi \bar{p}(\phi_1)$
$\tau.1$	$\neg \phi_1$	b_τ^ψ	a_τ^ψ
$\tau.1$	$\forall x \phi_1$	a_τ^ψ	b_τ^ψ
$\tau.1$	$\exists x \phi_1$	a_τ^ψ	b_τ^ψ
ϵ	ψ	1	0

Table 4: Calculating the coefficients

determine how often a particular subformula and its negation are duplicated in the course of a standard CNF translation. The coefficient a_π^ψ is the factor for the multiplication of $p(\psi|_\pi)$ in the CNF whereas the factor b_π^ψ is responsible for the multiplication of $\bar{p}(\psi|_\pi)$. The first column of Table 4 shows the form of π , the second column the form of ψ directly above position π (ψ itself if $\pi = \epsilon$). The next two columns demonstrate the corresponding recursive bottom-up calculations for a_π^ψ and b_π^ψ , respectively. Applied to our starting example formula $\psi = \phi_1 \vee \forall x \phi_2$ where we renamed position 2.1, i.e., the subformula ϕ_2 , the coefficients are $a_{2.1}^\psi = p(\phi_1)$ (Table 4, eighth, second and last row, first column) and $b_{2.1}^\psi = 0$ (eighth, second and last row, second column). Note that a_π^ψ (b_π^ψ) is always 0 if $pol(\psi, \pi) = -1$ ($pol(\psi, \pi) = 1$).

Using the notion of a coefficient, the previously stated condition can be reformulated as

$$a_\pi^\psi p(\phi) + b_\pi^\psi \bar{p}(\phi) \geq a_\pi^\psi + b_\pi^\psi + p(def(\pi, \psi, R))$$

where we still assume that $\phi = \psi|_{\pi}$. Note that, since ϕ is replaced by an atom, the coefficients a_{π}^{ψ} , b_{π}^{ψ} are multiplied by 1 in the renamed version. Depending on the polarity of $\psi|_{\pi}$ the inequality is equivalent to one of the three inequalities:

$$\begin{aligned} a_{\pi}^{\psi} p(\phi) &\geq a_{\pi}^{\psi} + p(\phi) && \text{if } \text{pol}(\psi, \pi) = 1 \\ b_{\pi}^{\psi} \bar{p}(\phi) &\geq b_{\pi}^{\psi} + \bar{p}(\phi) && \text{if } \text{pol}(\psi, \pi) = -1 \\ a_{\pi}^{\psi} p(\phi) + b_{\pi}^{\psi} \bar{p}(\phi) &\geq a_{\pi}^{\psi} + b_{\pi}^{\psi} + p(\phi) + \bar{p}(\phi) && \text{if } \text{pol}(\psi, \pi) = 0 \end{aligned}$$

By simple arithmetical transformations, we can group all occurrences of factors a_{π}^{ψ} , b_{π}^{ψ} and all occurrences of $p(\phi)$ and $\bar{p}(\phi)$, respectively:

$$\begin{aligned} (a_{\pi}^{\psi} - 1)(p(\phi) - 1) &\geq 1 && \text{if } \text{pol}(\psi, \pi) = 1 \\ (b_{\pi}^{\psi} - 1)(\bar{p}(\phi) - 1) &\geq 1 && \text{if } \text{pol}(\psi, \pi) = -1 \\ (a_{\pi}^{\psi} - 1)(p(\phi) - 1) + (b_{\pi}^{\psi} - 1)(\bar{p}(\phi) - 1) &\geq 2 && \text{if } \text{pol}(\psi, \pi) = 0 \end{aligned}$$

Let us abbreviate the product $(a_{\pi}^{\psi} - 1)(p(\phi) - 1)$ with p_a and $(b_{\pi}^{\psi} - 1)(\bar{p}(\phi) - 1)$ with p_b . Since neither p_a nor p_b can become negative, in any of the cases where they appear, the first inequality holds if $p_a \geq 1$, the second inequality holds if $p_b \geq 1$ and the third inequality holds if (i) $p_a \geq 2$ or (ii) $p_b \geq 2$ or (iii) $p_a \geq 1$ and $p_b \geq 1$. In order to check these conditions, it suffices to test whether the coefficients a_{π}^{ψ} , b_{π}^{ψ} and the number of clauses $p(\phi)$, $\bar{p}(\phi)$ are strictly greater than 1, 2 or 3, respectively. This can always be checked in linear time with respect to the size of ψ . The condition $p(\phi) > 1$ holds iff there exists a position π such that $\phi[\phi_1 \equiv \phi_2]_{\pi}$ or $\phi[\phi_1 \wedge \phi_2]_{\pi}$ and $\text{pol}(\phi, \pi) = 1$ or $\phi[\phi_1 \circ \phi_2]_{\pi}$ with $\text{pol}(\phi, \pi) = -1$ and $\circ \in \{\vee, \supset\}$. The computations for the boolean conditions $p(\phi) > 2$ and $p(\phi) > 3$ are depicted in Table 5. We only considered the case of a conjunction and disjunction, since all other cases can be reduced to these cases. The computation of the conditions for \bar{p} works accordingly, following Table 3.

ψ	$p(\psi) > 2$
$\phi_1 \wedge \phi_2$	$p(\phi_1) > 1$ or $p(\phi_2) > 1$
$\phi_1 \vee \phi_2$	$p(\phi_i) > 2$ or $p(\phi_1) > 1$ and $p(\phi_2) > 1$

ψ	$p(\psi) > 3$
$\phi_1 \wedge \phi_2$	$p(\phi_i) > 2$
$\phi_1 \vee \phi_2$	$p(\phi_i) > 3$ or $[p(\phi_i) > 2$ and $p(\phi_j) > 1, i \neq j]$

Table 5: The Boolean Conditions for p

As for the factors, Table 6 shows how to compute $a_{\pi}^{\psi} > 1$ and, following Table 4, this can be extended to the other cases for the a factor and the corresponding conditions for the b factor.

π	$\psi _{\tau}$	$a_{\pi}^{\psi} > 1$
$\tau.i$	$\phi_1 \wedge \phi_2$	$a_{\tau}^{\psi} > 1$
$\tau.i$	$\phi_1 \vee \phi_2$	$a_{\tau}^{\psi} > 1$ or $p(\phi_j) > 1$ for some j

Table 6: The Boolean Conditions for a

Hence we turned a test that required the computation of exponentially growing functions into a boolean condition that does not require any arithmetic calculation at all.

4.2. THEOREM (Formula Renaming). *Formula Renaming preserves satisfiability and can be computed in polynomial time.*

In order to further reduce the number of eventually generated clauses it may still be useful to rename a formula, even if the above considerations do not apply. For example, renaming the formula $P_1 \vee (Q_1 \wedge Q_2)$ at position 2 results in three clauses, whereas a standard CNF translation of the original formula yields two clauses. This calculation also applies if this situation is repeated, as in

$$[P_1 \vee (Q_1 \wedge Q_2)] \wedge [P_2 \vee (Q_1 \wedge Q_2)] \wedge \dots [P_n \vee (Q_1 \wedge Q_2)]$$

where our renaming criterion does not apply. But now a simultaneous renaming of all occurrences $(Q_1 \wedge Q_2)$ may pay off. It results in $n + 2$ clauses whereas the standard CNF translation yields $2n$ clauses. Hence, it is useful to search for multiple occurrences of the same subformula. The problem here is to find an appropriate “equality” or “instance” relation between subformulae. In our example syntactic equality was sufficient to detect all such occurrences. In general, a matching process – probably with respect to the commutativity, associativity of some logical operators or even logical implication – may be needed to obtain a suitable renaming result. So we run here into a tradeoff between compact CNFs and computational complexity to achieve these CNFs. For example, if we slightly modify the above formula to

$$[P_1 \vee (Q_1(a_1) \wedge Q_2(a_1))] \wedge [P_2 \vee (Q_1(a_2) \wedge Q_2(a_2))] \wedge \dots [P_n \vee (Q_1(a_n) \wedge Q_2(a_n))]$$

syntactic equality is not sufficient to simultaneously rename all occurrences of conjuncts $(Q_1(a_i) \wedge Q_2(a_i))$. Here we need a generalization. We add the definition $\forall x, y (R(x, y) \supset (Q_1(x) \wedge Q_2(y)))$ and replace the i^{th} occurrence of the conjunct by $R(x, y)\{x \mapsto a_i, y \mapsto a_i\}$ where $\{x \mapsto a_i, y \mapsto a_i\}$ is the matcher between $(Q_1(x) \wedge Q_2(y))$ and $(Q_1(a_i) \wedge Q_2(a_i))$.

Repeated application of formula renamings prevents the explosion in the number of clauses in the course of a CNF transformation. Furthermore, it is well-known that the introduction of new predicate symbols that abbreviate formulae, can cause an exponential decrease in proof length of an afterwards found resolution proof. On the other hand, the introduction of renamings may also increase proof (search)

complexity. Consider a formula of the form $P \equiv Q \equiv P \equiv Q \dots$ with n occurrences of P (and Q) where we assume that \equiv binds to the left. The formula will be renamed for $n > 1$ where we introduce new propositional variables, linearly many in n , if we perform every renaming of a subformula that results in a smaller clause normal form with respect to our calculations. Hence, the number of truth assignments of the renamed formula grows exponentially in the size of the input formula. The problem is that the above formula is highly redundant since there are at most four different, non-redundant clauses in the sense of Section 3.7 with respect to two propositional variables. Redundancy is not considered by the calculations we developed in this section and hence the renaming method may produce worse results in this case. On the other hand, the standard CNF algorithm, we presented in Section 3 would generate exponentially many clauses after Section 3.6 which finally collapse to at most four clauses in Section 3.7. To prevent the intermediate exponential blowup, a much more sophisticated CNF procedure is needed. The point we want to make here is that formula renaming does not necessarily support proof search and that the design of a good CNF procedure is far from straightforward. Nevertheless, our experience shows that formula renaming, as we stated here, is a very useful mechanism (see Section 7 and Section 8).

Finally, it should be noted that after application of formula renamings and CNF generation, the “original” standard CNF can be reproduced by resolving on the literals that have been newly introduced in the renaming process. This may be an undesired behavior that can be inhibited by using ordered resolution with an appropriate ordering where literals built from the newly introduced predicate symbols are “small,” see [Weidenbach 2001] (Chapter 27 of this Handbook).

5. Skolemization

One of the most important parts in the process of clause normal form transformation is *Skolemization*. It is used to get rid of existentially quantified variables and that by replacing each such occurrence with a Skolem function application. This, ultimately, leads to formulae in which all quantifications are universal (see Section 3).

Usually, there are two kinds of Skolemization techniques mentioned in the relevant literature on automated theorem proving which we call *Inner* and *Outer Skolemization* in this paper. The two differ mainly in the choice of the arguments for the Skolem functions. An existential quantifier to be replaced by some Skolem function lies within the scope of some universally quantified variables but also has some free variables within its own scope. Outer Skolemization takes the former variables as arguments for Skolem functions, whereas Inner Skolemization utilizes the latter variables. For both kinds of Skolemization it has been shown to be sometimes valuable to initially transform the given problem into antiprenex normal form.¹

Here and in the sequel we assume uniqueness of quantified subformulae, i.e., no

¹A formula is said to be in *antiprenex normal form* if its quantifiers are moved inwards as far as possible. There is one exception, however, existential quantifiers are moved outwards over disjunctions.

variable symbol is bound twice. Evidently, this can always easily be achieved by a suitable variable renaming.

5.1. DEFINITION (Standard Skolemizations). Let ϕ be a sentence in negation normal form and let $\exists x \psi = \phi|_\pi$ for some $\pi \in \text{pos}(\phi)$. Moreover, let $\{\bar{y}_n\} = \text{free}(\exists x \psi)$ and $\{\bar{z}_m\} = \{z \mid \forall z \xi = \phi|_\tau, \pi > \tau\}$. Then from ϕ we can obtain

- $\phi[\pi/\psi\{x \mapsto f(\bar{y}_n)\}]$ by an *inner Skolemization step* (see also 3.5), and
- $\phi[\pi/\psi\{x \mapsto f(\bar{z}_m)\}]$ by an *outer Skolemization step*,

where f is a new n -place (m -place respectively) function symbol.

Standard Skolemizing a formula means to apply standard Skolemization steps until all existential quantifiers are eliminated. Although the final result is in general not logically equivalent to the original formula, we know that preserves satisfiability.

Not much effort had been spent until recently to improve these standard Skolemization techniques. Nevertheless, even Skolemization leaves possibilities for further improvements. In this article we want to present two such proposals, the optimized Skolemization and the strong Skolemization. Both of these two new Skolemization methods are generalizations of Inner Skolemization in the sense that their Skolemization result subsumes the one of Inner Skolemization. This often allows for considerable simplifications after generating normal forms for automated theorem proving and may thus lead to a reduction of both search space and proof length.

5.2. DEFINITION (Optimized Skolemization). Let ϕ be a sentence in negation normal form, let $\exists \bar{x}_k (\psi_1 \wedge \psi_2) = \phi|_\pi$ with $\{\bar{y}_n\} = \text{free}(\phi|_\pi)$, and let f_1, \dots, f_k be new (Skolem) function symbols. If $\models \phi \supset \forall \bar{y}_n \exists \bar{x}_k \psi_1$ then

$$\forall \bar{y}_n \psi_1 \{ \dots, x_i \mapsto f_i(\bar{y}_n), \dots \} \wedge \phi[\pi/\psi_2 \{ \dots, x_i \mapsto f_i(\bar{y}_n), \dots \}]$$

can be obtained by an *optimized Skolemization step* from ϕ .

5.3. LEMMA. *Optimized Skolemization preserves satisfiability.*

PROOF. According to our definition of satisfiability preservation from page 339 we have to show that a formula is satisfiable if and only if the result of an optimized Skolemization step on this formula is satisfiable. Let us first consider the direction from right to left, i.e., assume that $\mathcal{M} \models \forall \bar{y}_n \psi_1 \{ \dots, x_i \mapsto f_i(\bar{y}_n), \dots \} \wedge \phi[\pi/\psi_2 \{ \dots, x_i \mapsto f_i(\bar{y}_n), \dots \}]$. Then $\mathcal{M} \models \forall \bar{y}_n (\psi_2 \{ \dots, x_i \mapsto f_i(\bar{y}_n), \dots \} \supset \exists \bar{x}_k (\psi_1 \wedge \psi_2))$ and so $\mathcal{M} \models \phi$. Hence, if ϕ is unsatisfiable then the result of an optimized skolemization step must be unsatisfiable as well for otherwise, as the above shows, we would be able to find a model for ϕ .

For the other direction of the proof assume that $\mathcal{M} \models \phi$. We have to show that there exists an interpretation \mathcal{M}' that satisfies the result of an optimized skolemization step on ϕ . We do so by constructing such an interpretation; in fact we show that there exists a suitable \mathcal{M}' that differs from \mathcal{M} only in the interpretation of the new function symbols f_1, \dots, f_k , i.e., $\mathcal{M}' = \langle \mathcal{D}, \mathcal{I}', \nu \rangle$ where $\mathcal{M} = \langle \mathcal{D}, \mathcal{I}, \nu \rangle$ and \mathcal{I}' extends \mathcal{I} by suitable interpretations for f_1, \dots, f_k . Now, consider an arbitrary

sequence \bar{a}_n of domain values. If $\mathcal{M}[\bar{y}_n/\bar{a}_n] \models \exists \bar{x}_k (\psi_1 \wedge \psi_2)$ then we choose the f_i such that $\mathcal{I}'(f_i)(\bar{a}_n) = b_i$, where the b_i are witnesses for the above existentially quantified variables x_i . If, on the other hand, $\mathcal{M}[\bar{y}_n/\bar{a}_n] \not\models \exists \bar{x}_k (\psi_1 \wedge \psi_2)$ then we choose the f_i such that $\mathcal{I}'(f_i)(\bar{a}_n) = c_i$ where $\mathcal{M}[\bar{y}_n/\bar{a}_n, \bar{x}_k/\bar{c}_k] \models \psi_1$. Note that according to our assumption that $\models \phi \supset \forall \bar{y}_n \exists \bar{x}_k \psi_1$ such \bar{c}_k always exist. Now, the f_i have been constructed such that $\langle \mathcal{D}, \mathcal{I}', \nu \rangle \models \phi \wedge (\exists \bar{x}_k (\psi_1 \wedge \psi_2) \supset \psi_2 \{x_i \mapsto f_i(\bar{y}_n)\})$ and so $\langle \mathcal{D}, \mathcal{I}', \nu \rangle \models \phi[\pi/\psi_2 \{x_i \mapsto f_i(\bar{y}_n)\}]$. Moreover, by construction, $\langle \mathcal{D}, \mathcal{I}', \nu \rangle \models \forall \bar{y}_n \exists \bar{x}_k \psi_1 \wedge (\exists \bar{x}_k \psi_1 \supset \psi_1 \{x_i \mapsto f_i(\bar{y}_n)\})$ and therefore $\langle \mathcal{D}, \mathcal{I}', \nu \rangle \models \forall \bar{y}_n \psi_1 \{x_i \mapsto f_i(\bar{y}_n)\}$ and we are done. \square

Note that optimized Skolemization in general requires a theorem prover on its own in order to prove the preliminary condition $\models \phi \supset \forall \bar{y}_n \exists \bar{x}_k \psi_1$.

As an example consider the subformula

$$\forall x, y, z (R(x, y) \wedge R(x, z) \supset \exists u (R(y, u) \wedge R(z, u)))$$

of some sentence ϕ and assume that $\forall x \exists y R(x, y)$ is provable from ϕ . With standard Inner Skolemization we would obtain the clauses

$$\neg R(x, y) \vee \neg R(x, z) \vee R(y, f(y, z))$$

$$\neg R(x, y) \vee \neg R(x, z) \vee R(z, f(y, z))$$

With optimized Skolemization, however, we would end up with the clause set

$$R(y, f(y, z))$$

$$\neg R(x, y) \vee \neg R(x, z) \vee R(z, f(y, z))$$

which is definitely superior to the standard result. Intuitively, the effect of optimized Skolemization compared to standard (Inner) Skolemization is that some of the literals in the standard result are deleted.

Strong Skolemization differs from optimized Skolemization in many respects. First of all, just like standard (Inner) Skolemization, it is a local method, i.e., it applies only to the subformula under consideration and does not take the whole problem into account. Moreover, it does not require a theorem prover on its own to perform its task. Now, before we define what strong Skolemization is about, let us first introduce the notion of a *free variable splitting*.

5.4. DEFINITION (Free Variable Splitting). Let ϕ be a sentence in negation normal form and let $\phi|_\pi = \exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)$ be a subformula of ϕ . Moreover, let

$$\{\bar{z}_{1l_1}\} = \text{free}(\psi_1) \setminus \{\bar{x}_k\}$$

$$\{\bar{z}_{il_i}\} = (\text{free}(\psi_i) \setminus \bigcup_{1 \leq j < i} \text{free}(\psi_j)) \setminus \{\bar{x}_k\} \text{ for } 1 < i \leq n.$$

We call $\{\{\bar{z}_{1l_1}\}, \dots, \{\bar{z}_{nl_n}\}\}$ the *free variable splitting* of $\phi|_\pi$.

Intuitively, each $\{\bar{z}_{il_i}\}$ contains the free variables of ψ_i (without the existentially quantified \bar{x}_k) that did not yet occur in ψ_j , $j < i$. Such a free variable splitting has some interesting features. For instance, its elements are pairwise disjoint and their overall union covers exactly the free variables of the formula $\exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)$. An important property which we will make use of

in the sequel is the following: if $\langle \{\bar{z}_{1l_1}\}, \dots, \{\bar{z}_{nl_n}\} \rangle$ is a free variable splitting of $\exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)^2$ then $\langle \{\bar{z}_{1l_1}, \bar{z}_{2l_2}\}, \{\bar{z}_{3l_3}\}, \dots, \{\bar{z}_{nl_n}\} \rangle$ is a free variable splitting of $\exists \bar{v}_k (\psi_2 \wedge \dots \wedge \psi_n) \{x_i \mapsto f_i(\bar{z}_{1l_1}, \bar{v}_k)\}$ provided the variables \bar{v}_k are new to $\exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)$.

With this definition standard Inner Skolemization could be described as follows. Replace any occurrence of a subformula $\exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)$ with

$$\begin{aligned} & \psi_1 \{x_i \mapsto f_i(\bar{z}_{1l_1}, \dots, \bar{z}_{nl_n})\} \wedge \\ & \psi_2 \{x_i \mapsto f_i(\bar{z}_{1l_1}, \dots, \bar{z}_{nl_n})\} \wedge \\ & \quad \vdots \\ & \psi_n \{x_i \mapsto f_i(\bar{z}_{1l_1}, \dots, \bar{z}_{nl_n})\} \end{aligned}$$

where $\langle \{\bar{z}_{1l_1}\}, \dots, \{\bar{z}_{nl_n}\} \rangle$ is the free variable splitting of $\exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)$.

The effect of strong Skolemization lies in replacing some of these variable sequences with sequences of fresh universally quantified variables. The following definition specifies this.

5.5. DEFINITION (Strong Skolemization). Let ϕ be a sentence in negation normal form with $\phi|_\pi = \exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)$. A strong Skolemization step yields $\phi[\pi/\xi]$ with $\xi =$

$$\begin{aligned} & \forall \bar{w}_2, \bar{w}_3, \bar{w}_4, \dots, \bar{w}_n \psi_1 \{x_i \mapsto f_i(\bar{z}_1, \bar{w}_2, \bar{w}_3, \bar{w}_4, \dots, \bar{w}_n)\} \wedge \\ & \forall \bar{w}_3, \bar{w}_4, \dots, \bar{w}_n \psi_2 \{x_i \mapsto f_i(\bar{z}_1, \bar{z}_2, \bar{w}_3, \bar{w}_4, \dots, \bar{w}_n)\} \wedge \\ & \quad \vdots \\ & \forall \bar{w}_n \psi_{n-1} \{x_i \mapsto f_i(\bar{z}_1, \bar{z}_2, \dots, \bar{z}_{n-1}, \bar{w}_n)\} \wedge \\ & \psi_n \{x_i \mapsto f_i(\bar{z}_1, \bar{z}_2, \dots, \bar{z}_{n-1}, \bar{z}_n)\} \end{aligned}$$

where $\langle \{\bar{z}_1\}, \dots, \{\bar{z}_n\} \rangle$ is the free variable splitting of $\exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)$. Each variable sequence \bar{w}_i has length equal to the length of the variable sequence \bar{z}_i , and the $f_i, 1 \leq i \leq k$, are new (Skolem) function symbols.

In the proof that strong Skolemization behaves as desired we shall make use of some preliminary definitions as well as an important key lemma.

5.6. DEFINITION (Selector Functions, Projections). A *selector function* for a non-empty set $S \subseteq \mathcal{D}^k$ is a total function $h : \mathcal{D}^k \rightarrow \mathcal{D}^k$ such that its co-domain is just S , i.e., $\text{cdom}(h) = S$.

As usual, the *projection functions* proj_i allow us to refer to the i^{th} element of a given sequence, i.e., $\text{proj}_i(e_1, \dots, e_i, \dots, e_n) = e_i$.

5.7. LEMMA. If $\exists \bar{x}_k (\phi \wedge \psi)$ is satisfiable then $\forall \bar{w}_k \phi \{x_i \mapsto h_i(\bar{y}_n, \bar{w}_k)\} \wedge \exists \bar{v}_k \psi \{x_i \mapsto h_i(\bar{y}_n, \bar{v}_k)\}$ is satisfiable as well, where $\{\bar{y}_n\} = \text{free}(\phi) \setminus \{\bar{x}_k\}$.

²We assume that each of the ψ_i contains at least one of the variables from \bar{x}_k . Otherwise, such a ψ_i could be shifted out of the scope of the existential quantification (see also the simplification rules in Section 3.3).

PROOF. Let $\mathcal{M} = \langle \mathcal{D}, \mathcal{I}, \nu \rangle$ be an interpretation and define

$$S^\phi(\bar{b}_n) = \{\bar{a}_k \mid \mathcal{M}[\bar{y}_n/\bar{b}_n, \bar{x}_k/\bar{a}_k] \models \phi\}$$

Since $\mathcal{M} \models \exists \bar{x}_k (\phi \wedge \psi)$, we know that $S^\phi(\nu(\bar{y}_n)) \neq \emptyset$, where $\nu(\bar{y}_n)$ abbreviates the sequence $\nu(y_1), \dots, \nu(y_n)$. Thus we know that

$$\mathcal{M}[\bar{x}_k/\bar{a}_k] \models \phi \quad \text{for all } \bar{a}_k \in S^\phi(\nu(\bar{y}_n)) \text{ and}$$

$$\mathcal{M}[\bar{x}_k/\bar{a}_k] \models \psi \quad \text{for some } \bar{a}_k \in S^\phi(\nu(\bar{y}_n)).$$

Now, for arbitrary $\bar{c}_n \in \mathcal{D}^n$, let $h^\phi(\bar{c}_n): \mathcal{D}^k \rightarrow \mathcal{D}^k$ be a *selector function* for $S^\phi(\bar{c}_n) \subseteq \mathcal{D}^k$. Then

$$\mathcal{M}[\bar{x}_k/h^\phi(\nu(\bar{y}_n))(\bar{a}_k)] \models \phi \quad \text{for all } \bar{a}_k \in \mathcal{D}^k \text{ and}$$

$$\mathcal{M}[\bar{x}_k/h^\phi(\nu(\bar{y}_n))(\bar{a}_k)] \models \psi \quad \text{for some } \bar{a}_k \in \mathcal{D}^k$$

$h^\phi(\nu(\bar{y}_n))$ returns sequences of domain elements. Thus, in order to represent a substitute for each of the x_i , we have to refer to the respective projections on these sequences, i.e.,

$$\mathcal{M}[\dots, x_i/proj_i(h^\phi(\nu(\bar{y}_n))(\bar{a}_k)), \dots] \models \phi \quad \text{for all } \bar{a}_k \in \mathcal{D}^k \text{ and}$$

$$\mathcal{M}[\dots, x_i/proj_i(h^\phi(\nu(\bar{y}_n))(\bar{a}_k)), \dots] \models \psi \quad \text{for some } \bar{a}_k \in \mathcal{D}^k$$

This immediately leads to a suitable interpretation for the new function symbols h_1, \dots, h_k which extend \mathcal{M} to \mathcal{M}' by

$$\mathcal{M}'(h_i)(\bar{b}_n, \bar{a}_k) = proj_i(h^\phi(\bar{b}_n)(\bar{a}_k)).$$

With this we get

$$\mathcal{M}'[\bar{w}_k/\bar{a}_k] \models \phi\{x_i \mapsto h_i(\bar{y}_n, \bar{w}_k)\} \quad \text{for all } \bar{a}_k \in \mathcal{D}^k \text{ and}$$

$$\mathcal{M}'[\bar{v}_k/\bar{a}_k] \models \psi\{x_i \mapsto h_i(\bar{y}_n, \bar{v}_k)\} \quad \text{for some } \bar{a}_k \in \mathcal{D}^k$$

which finally can be simplified to

$$\mathcal{M}' \models \forall \bar{w}_k \phi\{x_i \mapsto h_i(\bar{y}_n, \bar{w}_k)\} \wedge \exists \bar{v}_k \psi\{x_i \mapsto h_i(\bar{y}_n, \bar{v}_k)\}$$

and this completes the proof. □

5.8. LEMMA. *Strong Skolemization preserves satisfiability.*

PROOF. We have to show that a formula ϕ is satisfiable if and only if the result of a strong Skolemization step on ϕ is satisfiable. The direction from right to left follows immediately from the fact that strong Skolemization generalizes standard Inner Skolemization. As for the other direction, assume that $\mathcal{M} \models \phi$ with $\phi|_\pi = \exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)$. Without loss of generality we assume that each of the

ϕ_i contains at least one of the variables from \bar{x}_k . We prove by induction on the structure of ϕ that there exists an interpretation \mathcal{M}' that differs from \mathcal{M} only in a suitable interpretation for the new Skolem function symbols f_1, \dots, f_k and that satisfies the strong Skolemization result on the indicated subformula.

The base case (where $\phi = \exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)$) is in fact the crucial part of the proof and requires another induction; this time on the number of conjuncts in the subformula under consideration (n in the formula above).

For the case $n = 1$ strong Skolemization does not differ from standard Inner Skolemization and we are done. For the induction step consider for $n > 1$ that $\mathcal{M} \models \exists \bar{x}_k (\psi_1 \wedge \dots \wedge \psi_n)$. Then, by Lemma 5.7, we know that there exists an interpretation \mathcal{M}' such that

$$\mathcal{M}' \models \forall \bar{w}_k \psi_1 \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{w}_k)\} \wedge \exists \bar{v}_k (\psi_2 \wedge \dots \wedge \psi_n) \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{v}_k)\}$$

where \mathcal{M}' is like \mathcal{M} except for the additional interpretation for the new function symbols h_i . The second conjunct of this new formula is a sequence of existential quantifiers followed by a conjunction that consists of $n - 1$ conjuncts. This means that we can apply the induction hypothesis on this subformula. Now, in order to apply the induction hypothesis on $\exists \bar{v}_k (\psi_2 \wedge \dots \wedge \psi_n) \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{v}_k)\}$ recall that the corresponding free variable splitting is given by $\{\{\bar{z}_{1l_1}, \bar{z}_{2l_2}\}, \{\bar{z}_{3l_3}\}, \dots, \{\bar{z}_{nl_n}\}\}$. Thus, an application of strong Skolemization leads to the satisfiability of

$$\begin{aligned} & \forall \bar{w}_k \quad \psi_1 \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{w}_k)\} \wedge \\ & \forall \bar{w}_{3l_3}, \dots, \bar{w}_{nl_n} \quad \psi_2 \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{v}_k)\} \{v_j \mapsto g_j(\bar{z}_{1l_1}, \bar{z}_{2l_2}, \bar{w}_{3l_3}, \dots, \bar{w}_{nl_n})\} \wedge \\ & \forall \bar{w}_{4l_4}, \dots, \bar{w}_{nl_n} \quad \psi_3 \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{v}_k)\} \{v_j \mapsto g_j(\bar{z}_{1l_1}, \dots, \bar{z}_{3l_3}, \bar{w}_{4l_4}, \dots, \bar{w}_{nl_n})\} \wedge \\ & \quad \vdots \\ & \forall \bar{w}_{nl_n} \quad \psi_{n-1} \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{v}_k)\} \{v_j \mapsto g_j(\bar{z}_{1l_1}, \dots, \bar{z}_{n-1l_{n-1}}, \bar{w}_{nl_n})\} \wedge \\ & \quad \psi_n \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{v}_k)\} \{v_j \mapsto g_j(\bar{z}_{1l_1}, \bar{z}_{2l_2}, \dots, \bar{z}_{nl_n})\} \end{aligned}$$

Instantiating the variables w_i in the first conjunct with $g_i(\bar{z}_{1l_1}, \bar{u}_{2l_2}, \dots, \bar{u}_{nl_n})$ – where the elements in each $\bar{u}_{j_l_j}$ are fresh universally quantified variables – then reveals the satisfiability of

$$\begin{aligned} & \forall \bar{u}_{2l_2}, \dots, \bar{u}_{nl_n} \quad \psi_1 \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{v}_k)\} \{v_j \mapsto g_j(\bar{z}_{1l_1}, \bar{u}_{2l_2}, \dots, \bar{u}_{nl_n})\} \wedge \\ & \forall \bar{w}_{3l_3}, \dots, \bar{w}_{nl_n} \quad \psi_2 \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{v}_k)\} \{v_j \mapsto g_j(\bar{z}_{1l_1}, \bar{z}_{2l_2}, \bar{w}_{3l_3}, \dots, \bar{w}_{nl_n})\} \wedge \\ & \quad \vdots \\ & \forall \bar{w}_{nl_n} \quad \psi_{n-1} \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{v}_k)\} \{v_j \mapsto g_j(\bar{z}_{1l_1}, \dots, \bar{z}_{n-1l_{n-1}}, \bar{w}_{nl_n})\} \wedge \\ & \quad \psi_n \{x_i \mapsto h_i(\bar{z}_{1l_1}, \bar{v}_k)\} \{v_j \mapsto g_j(\bar{z}_{1l_1}, \bar{z}_{2l_2}, \dots, \bar{z}_{nl_n})\} \end{aligned}$$

Now let $f_i(\bar{u}_{1l_1}, \dots, \bar{u}_{nl_n}) = h_i(\bar{u}_{1l_1}, g_1(\bar{u}_{1l_1}, \dots, \bar{u}_{nl_n}), \dots, g_k(\bar{u}_{1l_1}, \dots, \bar{u}_{nl_n}))$ for arbitrary $\bar{u}_{1l_1}, \dots, \bar{u}_{nl_n}$. This then leads to the satisfiability of

$$\begin{aligned}
& \forall \overline{w}_{2l_2}, \overline{w}_{3l_3}, \dots, \overline{w}_{nl_n} \quad \psi_1 \{x_i \mapsto f_i(\overline{z}_{1l_1}, \overline{w}_{2l_2}, \dots, \overline{w}_{nl_n})\} \wedge \\
& \quad \forall \overline{w}_{3l_3}, \dots, \overline{w}_{nl_n} \quad \psi_2 \{x_i \mapsto f_i(\overline{z}_{1l_1}, \overline{z}_{2l_2}, \overline{w}_{3l_3}, \dots, \overline{w}_{nl_n})\} \wedge \\
& \quad \vdots \\
& \quad \forall \overline{w}_{nl_n} \quad \psi_{n-1} \{x_i \mapsto f_i(\overline{z}_{1l_1}, \dots, \overline{z}_{n-1l_{n-1}}, \overline{w}_{nl_n})\} \wedge \\
& \quad \psi_n \{x_i \mapsto f_i(\overline{z}_{1l_1}, \overline{z}_{2l_2}, \dots, \overline{z}_{nl_n})\}
\end{aligned}$$

and we are done with the base case of the outer induction proof.

The induction steps then follow trivially from the induction hypothesis. \square

Evidently, the strong Skolemization result subsumes what we could possibly get from standard Inner Skolemization (simply instantiate the \overline{w}_i with the corresponding \overline{z}_i). Intuitively, the effect of strong Skolemization compared to standard Inner Skolemization is that some of the arguments of the new Skolem functions are replaced with new, universally quantified variables.

Let us again have a look at the example

$$\forall x, y, z (R(x, y) \wedge R(x, z) \supset \exists u (R(y, u) \wedge R(z, u))).$$

Strong Skolemization would lead us to the clause set

$$\begin{aligned}
& \neg R(x, y) \vee \neg R(x, z) \vee R(y, f(y, w)) \\
& \neg R(x, y) \vee \neg R(x, z) \vee R(z, f(y, z))
\end{aligned}$$

which is almost identical to the standard Skolemization outcome, with one exception however, the new variable w in the first clause. In fact, this tiny change allows us to perform a condensation step on the first two literals, so that we finally end up with

$$\begin{aligned}
& \neg R(x, y) \vee R(y, f(y, w)) \\
& \neg R(x, y) \vee \neg R(x, z) \vee R(z, f(y, z))
\end{aligned}$$

This result is not quite as strong as the one we obtained from optimized Skolemization. But notice that it did not require to prove the goal $\forall x \exists y R(x, y)$ which might actually be hard to prove, provided it is at all provable. In fact, proving such preliminary lemmata can be arbitrarily complicated and so, in case of optimized Skolemization, suitable restrictions are necessary to ensure that such intermediate steps ultimately terminate.

Now recall the example from above. It shows the superiority of both strong and optimized Skolemization over standard Skolemization. Also, it suggests to prefer optimized Skolemization provided the intermediate goal $\forall y, z \exists u R(y, u)$ is (more or less easily) provable. For this reason, it might be argued to first perform an optimized Skolemization step and only if this was not successful to proceed with a strong Skolemization step. Such a heuristics has shown to be sensible in practice, although there are examples where an opposite direction would be more effective. For instance, consider the formula $\forall x, y (R(x, y) \supset \exists z (P(z) \wedge Q(x, z) \wedge S(x, y, z)))$ and suppose that the relation R is provably non-empty. Standard Skolemization

would then lead us to the clause set

$$\begin{aligned} &\neg R(x, y) \vee P(f(x, y)) \\ &\neg R(x, y) \vee Q(x, f(x, y)) \\ &\neg R(x, y) \vee S(x, y, f(x, y)) \end{aligned}$$

whereas strong Skolemization would come up with

$$\begin{aligned} &\neg R(x, y) \vee P(f(v, w)) \\ &\neg R(x, y) \vee Q(x, f(x, w)) \\ &\neg R(x, y) \vee S(x, y, f(x, y)). \end{aligned}$$

Because of the assumption that R is provably non-empty, we eventually might be able to derive $P(f(v, w))$ from the first clause.

In case of optimized Skolemization the attempt to prove that $\exists z P(z)$ would obviously be successful and so we would finally end up with the clause set

$$\begin{aligned} &P(f(x, y)) \\ &\neg R(x, y) \vee Q(x, f(x, y)) \\ &\neg R(x, y) \vee S(x, y, f(x, y)) \end{aligned}$$

which is a little less general than what we obtained from strong Skolemization (note the fresh variable w in the literal $Q(x, f(x, w))$ of the strong Skolemization result).

Thus there are examples which suggest to first try an optimized Skolemization step and then a strong Skolemization step (if the former was not successful in proving any of the intermediate goals), but also there are examples which propose the other way round. There is some evidence that a more tight combination of the two approaches is possible and valuable.

6. Simplification

Depending on the application area there usually exists a bunch of simplification techniques specific to that area. Often these techniques become available by particular semantic properties of the considered domain. For example, in a program verification context, syntactically different constructor terms generating a data type are usually interpreted by different domain elements. This implies that the domain of any model contains at least as many elements as there are constructor terms. This information can be exploited by simplification techniques (see Section 6.1) which falsify formulae that only hold in a domain of smaller size. As this example already suggests, advanced simplification techniques are usually so closely related to their specific domain that there cannot be a complete set of such techniques. Furthermore, such a presentation would not even be useful, since these techniques are often completely useless in other contexts. In this section we present three formula simplification techniques in the above sense. They should be considered as mere examples to motivate the investigation of such techniques in domains of interest.

6.1. Equality

The equality relation \approx is an important relation in automated reasoning. It is interpreted as the identity on the domain $\mathcal{I}(\approx) = \{(a, a) \mid a \in \mathcal{D}\}$ and serves, for example, as a means to encode the behavior of functions. The equality relation can be directly, finitely axiomatized in first-order logic. However, it turns out that the generated axioms result in a proliferation of the search space of an afterwards applied automated reasoning system. Therefore, it is now commonly agreed that equality must be supported by specific reasoning facilities [Nieuwenhuis and Rubio 2001] (Chapter 7 of this volume). Here we want to show that already on the formula level, the equality relation can be employed for simplifications. The two simplifications we suggest should be considered as examples for such simplifications. There are many more of such simplifications possible. However, their usefulness highly depends on the respective application domain. We therefore concentrate on the two examples below.

$$\boxed{\begin{array}{ll} \exists x [x \approx t \wedge \psi] & \rightarrow \quad \psi\{x \mapsto t\} \quad \text{if } x \notin \text{vars}(t) \\ \forall x [x \approx t \supset \psi] & \rightarrow \quad \psi\{x \mapsto t\} \quad \text{if } x \notin \text{vars}(t) \end{array}}$$

The rules terminate, since they reduce the number of equational atoms occurring in a formula. Furthermore, both rules are equivalence preserving.

6.1. LEMMA. *For any term t , variable x ($x \notin \text{vars}(t)$) and any formula ψ ,*

$$\begin{array}{ll} \mathcal{M} \models \exists x [x \approx t \wedge \psi] & \text{iff } \mathcal{M} \models \psi\{x \mapsto t\} \\ \mathcal{M} \models \forall x [x \approx t \supset \psi] & \text{iff } \mathcal{M} \models \psi\{x \mapsto t\} \end{array}$$

PROOF. We only show the first equivalence. The proof for the second is analogous. So, let us assume that $\mathcal{M} \models \exists x [x \approx t \wedge \psi]$. Then, by definition, there is some $a \in \mathcal{D}$ such that $\mathcal{M}[x/a] \models x \approx t$ and $\mathcal{M}[x/a] \models \psi$. The first part holds if we select an a with $a = \mathcal{M}(t)$. This can always be done, because $x \notin \text{vars}(t)$. Using $\mathcal{M}[x/a] \models \psi$ and an inductive argument on the structure of ψ , this implies $\mathcal{M} \models \psi\{x \mapsto t\}$. On the other hand, assume $\mathcal{M} \models \psi\{x \mapsto t\}$. Since $x \notin \text{vars}(t)$, the formula $\exists x x \approx t$ is valid witness the assignment $[x/\mathcal{M}(t)]$. We conclude $\mathcal{M} \models \exists x [x \approx t \wedge \psi]$ by an inductive argument on the structure of ψ . \square

Note that the above two rules together with the rules of Section 3.1 and the rule

$$\boxed{t \approx t \rightarrow \top}$$

are already strong enough to reduce all equational axioms mentioned above to \top . For example, a congruence axiom for some function f with arity two can be reduced as follows

$$\begin{array}{c} \forall x, y, z [x \approx y \supset f(x, z) \approx f(y, z)] \\ \downarrow \\ \forall x, y, z [f(y, z) \approx f(y, z)] \\ \downarrow \\ \top \end{array}$$

where we omitted some intermediate steps. The next set of simplification rules exploits properties of the domain of any interpretation satisfying the formula

$\psi[\forall y t \not\approx y]_\pi \rightarrow \psi[\pi/\perp]$	if $y \notin \text{vars}(t)$
$\psi[\forall y t \approx y]_\pi \rightarrow \psi[\pi/\perp]$	if $y \notin \text{vars}(t)$ and $\mathcal{M} \models \psi$ implies $ \mathcal{D} > 1$
$\psi[\forall y t \approx y]_\pi \rightarrow \psi[\pi/\top]$	if $y \notin \text{vars}(t)$ and $\mathcal{M} \models \psi$ implies $ \mathcal{D} = 1$
$\psi[\exists y t \approx y]_\pi \rightarrow \psi[\pi/\top]$	if $y \notin \text{vars}(t)$
$\psi[\exists y t \not\approx y]_\pi \rightarrow \psi[\pi/\top]$	if $y \notin \text{vars}(t)$ and $\mathcal{M} \models \psi$ implies $ \mathcal{D} > 1$
$\psi[\exists y t \not\approx y]_\pi \rightarrow \psi[\pi/\perp]$	if $y \notin \text{vars}(t)$ and $\mathcal{M} \models \psi$ implies $ \mathcal{D} = 1$

Since the rules introduce the logical constants \perp , \top they should again be combined with the rules of Section 3.1. Moreover, explicit or implicit miniscoping (Section 3.3) may be necessary to achieve a suitable form for the above rules. The rules terminate, because the right hand side of each rule contains less symbols than its left hand side. The rules are equivalence preserving, e.g., for any domain \mathcal{D} with $|\mathcal{D}| > 1$ the formula $\exists y t \not\approx y$ is obviously true and the formula $\forall y t \approx y$ is obviously false if $y \notin \text{vars}(t)$.

Four of the six transformation rules require cardinality properties of domains of models to be known. These are of course not decidable in general. For many applications, however, such properties are inherently given or can easily, even syntactically, be detected. For example, if some formula ψ contains as a conjunct the subformula $t \not\approx s$, then any model satisfying ψ has a domain with at least two elements.

6.2. Atom Definitions

An atom definition is a sentence of the form

$$\forall x_1, \dots, x_m [P(t_1, \dots, t_n) \equiv \psi]$$

where $\text{free}(P(t_1, \dots, t_n)) = \{x_1, \dots, x_m\}$. As we are only interested in sentences this implies $\text{free}(\psi) \subseteq \text{free}(P(t_1, \dots, t_n))$. In case $P(t_1, \dots, t_n)$ has the form $P(x_1, \dots, x_n)$ and the predicate P does not occur in ψ , such a definition may be used to completely eliminate the predicate P and the definition from a formula.

$\begin{aligned} \phi[P(s_1, \dots, s_n)]_\pi \wedge \theta &\rightarrow \phi[\pi/\psi\sigma] \wedge \theta \\ \text{if } \theta &\text{ implies the atom definition } \forall x_1, \dots, x_m [P(t_1, \dots, t_n) \equiv \psi] \\ \text{and } P(t_1, \dots, t_n)\sigma &= P(s_1, \dots, s_n) \end{aligned}$
--

The transformation is equivalence preserving, but in general exhaustive application of the reduction rule may not terminate, e.g., if ψ contains the predicate P . If ψ does not contain P and all t_i are different variables, then exhaustive application of the rule using a single atom definition terminates. The usefulness of an application of the rule highly depends on the context and the atom definition. There are application domains, e.g., mathematical theories or software verification where the expansion of atom definitions can be indispensable for the successful application of an automated reasoning system.

The notion of an atom definition can be generalized to sentences of the form

$$\forall x_1, \dots, x_m [\phi \supset (P(t_1, \dots, t_n) \equiv \psi)]$$

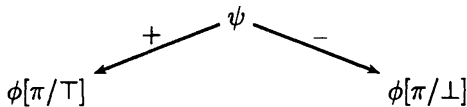
where we again require $\text{free}(P(t_1, \dots, t_n)) = \{x_1, \dots, x_m\}$ implying $\text{free}(\psi) \subseteq \text{free}(P(t_1, \dots, t_n))$ and $\text{free}(\phi) \subseteq \text{free}(P(t_1, \dots, t_n))$. For an application of this extended form, the formula ϕ must be valid in the context of the atom to be replaced. Of course, the applicability of such a definition is undecidable, in general. But there are domains where such definitions frequently occur and their applicability can be decided. For example, in a software verification domain, the formula ϕ may express sort/type restrictions that are usually decidable in that context.

As a transformation, the expansion of atom definitions and formula renaming (Section 4) seem to be inverse to each other. However, recall that we only apply formula renaming if this results in a smaller CNF and that the expansion of atom definitions makes no assumptions on the size. Its applicability should be decided on the basis of specific application domains or problems. This can lead to a combination where atom definitions are considered first and then the resulting formulae are considered for formula renamings, as it is done in SPASS, see [Weidenbach 2001] (Chapter 27 of this Handbook).

6.3. Binary Decision Diagrams

Another way to simplify formulae to obtain small clause normal forms is based on so-called *binary decision diagrams*, or *BDDs*, see [Clarke and Schlingloff 2001] (Chapter 24 of this Handbook). Such BDDs usually serve as rather compact representations of propositional formulae, they also gain more and more attention in the case of first-order logic, though.

The main idea behind BDDs is as follows. Given an arbitrary propositional formula $\phi[\psi]_\pi$, i.e., ψ is the sub-formula of ϕ at position π , we know that ϕ is equivalent to the conjunction of $\psi \supset \phi[\pi/\top]$ and $\neg\psi \supset \phi[\pi/\perp]$. This fact is often illustrated by a diagram of the following kind:



The $+$ -arrow indicates the positive case, whereas the other one is responsible for the negative case. The above diagram is thus to be read as:

$$\text{if } \psi \text{ then } \phi[\pi/\top] \text{ else } \phi[\pi/\perp].$$

A formula is to be transformed such that each of its atomic and each of its quantified sub-formulae is split as indicated above. This way the bottom leaves will be just \top or \perp . After that specific simplification rules as for example

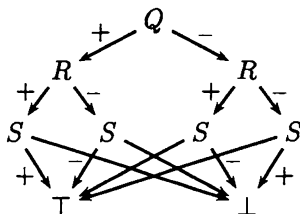
$$\text{if } \chi \text{ then } \xi \text{ else } \xi \rightarrow \xi$$

are exhaustively applied. Note that, given an ordering on the atoms and performing the above transformation with respect to this ordering, reduced BDDs form a unique

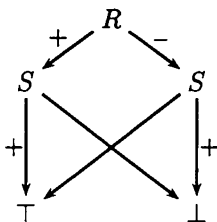
normal form for propositional logic. In particular, each valid (unsatisfiable) formula gets thus transformed into the unique reduced and ordered BDD \top (\perp respectively). As a simple example consider the propositional formula

$$(Q \equiv R) \equiv (Q \equiv S)$$

A corresponding BDD could be



Now note that the two sub-graphs below the node labeled Q are identical, thus we may simplify the graph to



from which we can immediately read off the clause normal form as

$$(\neg R \vee S) \wedge (R \vee \neg S).$$

This approach can be generalized to the first-order level. Then it not only allows us to detect simplifications as the one described above but also helps to detect *false dependencies* of existentially quantified variables. In this case these approaches follows a similar aim as the special Skolemization methods described in Section 5.

7. Bibliographic Notes

Eliminating equivalences depending on their polarity was first suggested by Henschen, Lusk, Overbeek, Smith, Veroff, Winker and Wos [1980] in order to solve Peter Andrew's famous "Challenge Problem 1" [Andrews 1979].

The renaming idea goes back to Tseitin [1968] who showed that the introduction of new propositional symbols can result in exponentially shorter proofs. In the context of CNF translation important contributions have been made by Eder [1992], Plaisted and Greenbaum [1986] and Boy de la Tour [1992]. These approaches differ in their respective criteria to decide which subformulae are to be replaced by new predicates. Whereas Plaisted and Greenbaum basically suggested to rename all subformulae up to literals, Boy de la Tour only replaces a subformula if this would decrease the number of eventually generated clauses. Plaisted and Greenbaum [1986]

also discuss the generalized formula renaming suggested at the end of Section 4. Since our goal is to few clauses, we followed the approach of Boy de la Tour [1992] where we presented his ideas using the notion of positions [Rock 1995]. Furthermore, we showed how to turn the approach into an efficient algorithm [Nonnengart, Rock and Weidenbach 1998]. This paper also contains an impressive experiment demonstrating the benefits of formula renaming and advanced Skolemization techniques. Boy de la Tour [1992] also proved that if a formula is checked top-down for possible renamings and does not contain equivalence symbols, then the renaming criterion in Section 4 produces an optimal³ CNF among all renamings. In this case Boy de la Tour [1992] showed that we can even strengthen the greater equal test for a renaming to take place to a strictly greater test⁴ and still obtain an optimal renaming. If an input formula contains equivalence symbols, the number of clauses of a CNF of a completely renamed formula grows linearly in the size of the input formula. As an upper bound for the number of symbols a CNF of a completely renamed formula ψ we obtain $|\psi|^3$ in case of the approach presented here and we obtain $|\psi|^2$ if the renaming criterion of Plaisted and Greenbaum [1986] is applied. Of course, all results abstract from potential effects that may be caused by redundant formulae (see page 352).

As for Skolemization issues, [Skolem 1920] is certainly worth a look at and that not only for historical reasons. More recent, though still classical references on this area can be found in [Andrews 1981, Loveland 1978, Chang and Lee 1973]. These cover essentially the standard approaches presented in this article. Refinements such as antiprenex transformation and miniscoping are described in [Egly 1994]. More information on the special Skolemization techniques introduced in this article, i.e., on strong as well as on optimized Skolemization, can be found in [Ohlbach and Weidenbach 1995, Nonnengart 1996, Nonnengart et al. 1998]. The reader who is rather interested in proof complexity issues is referred to the work of Baaz and Leitsch [1994] for an overview. Goubault [1995] introduces an approach of utilizing BDDs for the reduction of first-order formulae. This method is particularly interesting for it directly combines the reduction of BDDs with Skolemization. For the application of BDDs in first-order theorem proving Goubault and Posegga [1994] introduced a BDD-based approach and compared it with semantic tableaux.

Several techniques are not touched in this paper that seem to be valuable for further improvements. This includes, for example, potential reuse of Skolem functions and/or predicate symbols introduced by renaming, as described by Egly and Rath [1995].

8. Implementation Notes

Our considerations in Section 4 are motivated by the implementation of SPASS ([Weidenbach, Afshordel, Brahm, Cohrs, Engel, Keen, Theobalt and Topic 1999], see also [Weidenbach 2001] (Chapter 27 of this Handbook). Combining the renam-

³With respect to the number of finally generated clauses.

⁴See the disequations on page 348 and page 350 ff.

ing criteria developed in Section 4 with a top down traversal of the input formula provides the basis for an efficient implementation. As for data structures, it is important that the direct arguments of a formula as well as its direct super term can be efficiently accessed. A large number of experiments gave rise to the proposition that the renaming technique suggested here is in general superior to other renaming techniques and also to standard CNF translation [Nonnengart et al. 1998]. Concerning the renaming of multiple formula occurrences we mentioned on page 351, our experience shows that the following definition of matching is a good compromise between precision and complexity. We simply say that a formula ϕ is an instance of a formula ψ , if there exists a substitution σ with $\psi\sigma = \phi$.⁵ This can be checked in linear time with respect to the size of ψ and ϕ and furthermore, it allows us to use the size of a formula as a prefilter for this condition. Hence, it is useful to store formulae together with their size.

Acknowledgments

We want to thank Thierry Boy de la Tour, Uwe Egly and Enno Keen for their valuable comments.

Bibliography

- ANDREWS P. B. [1979], A challenge problem for automated theorem provers. The problem was posed at the fourth workshop on automated deduction, Austin.
- ANDREWS P. B. [1981], 'Theorem proving via general matings', *Journal of the ACM* **28**(2), 193–214.
- BAAZ M., EGLY U. AND LEITSCH A. [2001], Normal form transformations, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 5, pp. 273–333.
- BAAZ M. AND LEITSCH A. [1994], 'On skolemization and proof complexity', *Fundamenta Informaticae* **20**(4).
- BACHMAIR L. AND GANZINGER H. [2001], Resolution theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19–99.
- BOY DE LA TOUR T. [1992], 'An optimality result for clause form translation', *Journal of Symbolic Computation* **14**, 283–301.
- CHANG C.-L. AND LEE R. C.-T. [1973], *Symbolic Logic and Mechanical Theorem Proving*, Computer Science and Applied Mathematics, Academic Press.
- CLARKE E. AND SCHLINGLOFF H. [2001], Model checking, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 24, pp. 1635–1790.
- EDER E. [1992], *Relative Complexities of First Order Calculi*, Artificial Intelligence, Vieweg.
- EGLY U. [1994], On the value of antiprenexing, in 'Logic Programming and Automated Reasoning, 5th International Conference, LPAR'94', Vol. 822 of *LNAI*, Springer, pp. 69–83.
- EGLY U. AND RATH T. [1995], 'The halting problem: An automatically generated proof', *AAR Newsletter* **30**, 10–16.

⁵Recall the application of substitutions to formulae, page 339.

- FERMÜLLER C., LEITSCH A., HUSTADT U. AND TAMMET T. [2001], Resolution decision procedures, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 25, pp. 1791–1849.
- GOUBAULT J. [1995], 'A BDD-based simplification and skolemization procedure', *Logic Journal of the IGPL* 3(6), 827–855.
- GOUBAULT J. AND POSEGGA J. [1994], BDDs and automated deduction, in 'Proceedings of the 8th Int. Symp. on Methodologies for Intelligent Systems', Springer Verlag, LNAI.
- HENSCHEN L., LUSK E., OVERBEEK R., SMITH B., VEROFF R., WINKER S. AND WOS L. [1980], 'Challenge problem 1', *SIGART Newsletter* 72, 30–31.
- LOVELAND D. W. [1978], *Automated Theorem Proving: A Logical Basis*, Vol. 6 of *Fundamental Studies in Computer Science*, North-Holland.
- NIEUWENHUIS R. AND RUBIO A. [2001], Paramodulation-based theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 7, pp. 371–443.
- NONNENGART A. [1996], Strong skolemization, Research Report MPI-I-96-2-010, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany.
- NONNENGART A., ROCK G. AND WEIDENBACH C. [1998], On generating small clause normal forms, in '15th International Conference on Automated Deduction, CADE-15', Vol. 1421 of *LNAI*, Springer, pp. 397–411.
- OHLBACH H. J. AND WEIDENBACH C. [1995], 'A note on assumptions about skolem functions', *Journal of Automated Reasoning* 15(2), 267–275.
- PLAISTED D. A. AND GREENBAUM S. [1986], 'A structure-preserving clause form translation', *Journal of Symbolic Computation* 2, 293–304.
- ROCK G. [1995], Transformations of first-order formulae for automated reasoning, Diplomarbeit, Max-Planck-Institut für Informatik, Saarbrücken, Germany. Supervisors: H.J. Ohlbach, C. Weidenbach.
- SKOLEM T. [1920], 'Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theoreme über dichte Mengen', *Skrifter utgit av Videnskapsellkapet i Kristiania* 4, 4–36.
- TSEITIN G. [1968], On the complexity of derivations in propositional calculus, in A. Slisenko, ed., 'Studies in Constructive Mathematics and Mathematical Logic'. Reprinted in [Tseitin 1983].
- TSEITIN G. [1983], On the complexity of derivations in propositional calculus, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning: Classical Papers on Computational Logic', Vol. 2, Springer, pp. 466–483.
- WEIDENBACH C. [2001], Combining superposition, sorts and splitting, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 27, pp. 1965–2013.
- WEIDENBACH C., AFSHORDEL B., BRAHM U., COHRS C., ENGEL T., KEEN E., THEOBALT C. AND TOPIC D. [1999], System description: Spass version 1.0.0, in H. Ganzinger, ed., '16th International Conference on Automated Deduction, CADE-16', Vol. 1632 of *LNAI*, Springer, pp. 314–318.

Index

A

antiprenex normal form 352

B

bound variables 340

C

clause normal form 340

F

formula renaming 347

formula simplification 341

free variable splitting 354, 355

free variables 340

I

inner Skolemization 352–355

interpretation 338

M

miniscoping 343

N

negation normal form 341, 353

O

optimized Skolemization 353, 354, 359

outer Skolemization 352, 353

P

polarity dependent elimination 342

preservation of satisfiability . 339, 353, 356

projection functions 355

R

renaming

 formula 347

 variable 339

S

selector functions 355

Skolem functions 352

Skolemization 352

 inner 352–355

 optimized 353, 354, 359

 outer 352, 353

 standard 345, 353

 strong 353–355, 358

standard Skolemization 353

strong Skolemization 353–355, 358

substitution 339

V

variable renaming 344

Paramodulation-Based Theorem Proving

Robert Nieuwenhuis

Albert Rubio

SECOND READERS: Jieh Hsiang, Christopher Lynch, Michael Rusinowitch, and
Andrei Voronkov.

Contents

1	About this chapter	373
1.1	Paramodulation	373
1.2	Extending the unit equality case: ordered paramodulation	374
1.3	Redundancy and saturation	376
1.4	Computing with finite saturated sets	377
1.5	Paramodulation with constrained clauses	377
1.6	Paramodulation with built-in equational theories	379
1.7	Basic paramodulation with built-in equational theories	380
2	Preliminaries	380
2.1	Terms and (rewrite) relations	381
2.2	Term orderings	382
2.3	Equality clauses and Herbrand interpretations	384
2.4	Constraints and constrained clauses	385
3	Paramodulation calculi	385
3.1	The model generation method	386
3.2	Non-equality predicates	390
3.3	Clauses with variables	391
3.4	Completeness without constraint inheritance	393
3.5	General clauses	394
3.6	Selection of negative equations	396
3.7	Merging paramodulation and perfect models	397
4	Saturation procedures	399
4.1	Redundancy in practice	399
4.2	Redundancy and saturation in the ground case	401
4.3	Non-ground saturation procedures	405
4.4	More general notions of redundancy for clauses	407
4.5	Computing with saturated sets	409
4.6	Completion as an instance of saturation	411
4.7	Extended signatures	412
5	Paramodulation with constrained clauses	414
5.1	Equality constraint inheritance: basic strategies	414

HANDBOOK OF AUTOMATED REASONING

Edited by Alan Robinson and Andrei Voronkov

© 2001 Elsevier Science Publishers B.V. All rights reserved

5.2	Ordering constraint inheritance	417
5.3	Basic paramodulation	417
5.4	Saturation for constrained clauses	418
5.5	General constrained clauses	420
6	Paramodulation with built-in equational theories	421
6.1	E-compatible reduction orderings	421
6.2	Paramodulation modulo associativity and commutativity	423
6.3	Constraint inheritance and built-in theories	424
7	Symbolic constraint solving	425
7.1	Ordering constraint solving	425
8	Extensions	427
8.1	Paramodulation-based answer computation	427
8.2	Paramodulation-based decidability and complexity results	428
9	Perspectives	429
9.1	Basicness and redundancy	429
9.2	Orderings	430
9.3	Constraint solving	430
9.4	Indexing data structures	431
9.5	More powerful redundancy notions	431
9.6	More global future research directions	432
	Bibliography	432
	Index	440

1. About this chapter

The aim of this chapter is to review the fundamental techniques in paramodulation-based theorem proving, presenting them in a uniform framework. We start with easier subcases and progressively include the different extensions. Since the objective is to obtain a concise overview of the current state of the art, some of the historical developments that are not essential for the current results are omitted (further historical remarks on paramodulation are given in [Degtyarev and Voronkov 2001a] (Chapter 10 of this Handbook).

In this first section, the main concepts are introduced in an informal way, with emphasis on their intuitive background. This is done to facilitate the reading of subsequent sections, where all these notions are formally defined and explained in detail, and some of the main results are proved.

1.1. Paramodulation

Paramodulation originated as a development of resolution [Robinson 1965], one of the main computational methods in first-order logic, see [Bachmair and Ganzinger 2001] (Chapter 2 of this Handbook). For improving resolution-based methods, the study of the equality predicate has been particularly important, since reasoning with equality is well-known to be of great importance in mathematics, logic, and computer science. Robinson [1965] showed that resolution together with factoring is *refutation complete*, that is, the empty clause will eventually be inferred by systematically enumerating all consequences of an unsatisfiable set of clauses by (*binary*) *resolution*:

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \quad \text{if } \sigma = mgu(A, B)$$

where $mgu(A, B)$ denotes a most general unifier of A and B , and *factoring*:

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \quad \text{if } \sigma = mgu(A, B)$$

For dealing with the equality predicate \simeq by resolution, one can specify it by means of the following *congruence axioms* \mathcal{E} :

$$\begin{array}{ll} \rightarrow x \simeq x & (\text{reflexivity}) \\ x \simeq y \rightarrow y \simeq x & (\text{symmetry}) \\ x \simeq y \wedge y \simeq z \rightarrow x \simeq z & (\text{transitivity}) \\ x_1 \simeq y_1 \wedge \dots \wedge x_n \simeq y_n \rightarrow f(x_1, \dots, x_n) \simeq f(y_1, \dots, y_n) & (\text{monotonicity-I}) \\ x_1 \simeq y_1 \wedge \dots \wedge x_n \simeq y_n \\ \wedge P(x_1, \dots, x_n) \rightarrow P(y_1, \dots, y_n) & (\text{monotonicity-II}) \end{array}$$

In fact the monotonicity axioms are axiom *schemes*: one *monotonicity-I* axiom is required for each non-constant n -ary function symbol f , and, similarly, one *monotonicity-II* axiom for each predicate symbol P . A set S of clauses is satisfiable in first-order logic with equality if, and only if, $S \cup \mathcal{E}$ is satisfiable in first-order logic without equality¹.

However, it is easy to see that resolution and factoring with \mathcal{E} tend to cause the generation of too many (mostly unnecessary) new clauses. Therefore, Robinson and Wos explored another possibility. They tried to avoid the need for specifying equality by treating it as part of the logical language, i.e., directly considering first-order logic with equality. This requires the design of dedicated inference rules, like *paramodulation* [Robinson and Wos 1969]:

$$\frac{C \vee s \simeq t \quad D}{(C \vee D[t]_p) \sigma} \quad \text{if } \sigma = mgu(s, D|_p)$$

where $D|_p$ is the subterm of D at position p , and $D[t]_p$ denotes the result of replacing in D this subterm by t . Paramodulation, together with resolution and factoring, was proved refutation complete, under the presence of the reflexivity axiom and certain tautologies called the *functional reflexivity* axioms

$$f(x_1, \dots, x_n) \simeq f(x_1, \dots, x_n)$$

for every n -ary function symbol f of the alphabet. Later on, Brand [Brand 1975] proved that the functional reflexivity axioms were unnecessary, as well as paramodulation *into variables*, that is, paramodulations where $D|_p$ is a variable. However, even under these restrictions, paramodulation is difficult to control: unless additional refinements are considered, it quickly produces a large amount of unnecessary clauses, expanding the search space excessively.

The strengths and weaknesses of paramodulation have led to fruitful theoretical and practical research on paramodulation-based theorem proving. Concerning the practical research, a large number of experiments with paramodulation have been performed at the Argonne group by Wos, Overbeek, Henschen and others (see, e.g., [Wos 1988, Wos 1996] for references), and especially by McCune with his provers Otter [McCune 1994] and EQP [McCune 1997a] and his recent automated proof of the Robbins conjecture [McCune 1997b, McCune 1997c]. Concerning the theoretical research, the main techniques are reviewed in this chapter, and some aspects of their implementation in practical provers is discussed.

1.2. Extending the unit equality case: ordered paramodulation

An important tool in paramodulation is the use of *term orderings* for restricting the number of inferences. Paramodulation is in fact based on Leibniz' law for replacement of equals by equals. Now the basic idea of *ordered* paramodulation is to

¹Note that there is no logical equivalence. First-order logic (FOL) with equality has more expressive power: for instance, in FOL with equality the clause $x \simeq a \vee x \simeq b$ expresses that the cardinality of models is at most two, which cannot be expressed in FOL without equality.

only perform replacements of *big* terms by *smaller* ones, with respect to the given ordering \succ .

This is precisely the idea of (ordered) *rewriting*. Let us consider now unit equations: we address *word problems* of the form $E \models u \simeq v$, where E is a set of equations and $u \simeq v$ is another equation. Assume that \succ is a *reduction ordering* on terms (see Section 2 for the precise definitions). A term t is rewritten in one step with an equation $l \simeq r$ (or, equivalently, $r \simeq l$) of E by replacing a subterm $l\sigma$ of t by $r\sigma$, for some substitution σ such that $l\sigma \succ r\sigma$. For example, let E consist of the equations $\text{plus}(0, x) \simeq x$ and $\text{plus}(s(x), y) \simeq s(\text{plus}(x, y))$. Denoting each step by \rightarrow_E (and assuming the steps agree with \succ), we have

$$\text{plus}(s(s(0)), s(0)) \rightarrow_E s(\text{plus}(s(0), s(0))) \rightarrow_E s(s(\text{plus}(0, s(0)))) \rightarrow_E s(s(s(0)))$$

This (ordered) rewrite relation terminates: starting from some finite term t , after a finite number of steps a *normal form* (i.e., a term that cannot be rewritten any further) is obtained.

Now let \rightarrow_E^* denote zero or more of these steps (i.e., \rightarrow_E^* is the reflexive-transitive closure of the relation \rightarrow_E). A set of equations E is called *confluent* w.r.t. the given \succ if, whenever $s \rightarrow_E^* u$ and $s \rightarrow_E^* v$, there is some t such that $u \rightarrow_E^* t$ and $v \rightarrow_E^* t$. It is not difficult to see that then every term has a unique normal form. Furthermore, rewriting is then a decision procedure for deduction in the theory of E , since $E \models s = t$ if, and only if, s and t have the same normal form².

The first instances of ordered paramodulation appeared in *Knuth-Bendix completion* [Knuth and Bendix 1970]. Roughly, a completion procedure attempts to transform a given set of equations into an equivalent confluent one. A crucial step of the transformation process is the computation of *critical pairs* between equations. A critical pair is an equation obtained by *superposition*, the restricted version of paramodulation in which inferences only involve left hand sides of possible rewrite steps, i.e., only the big terms (w.r.t. \succ) are considered. During the completion process equations are simplified by rewriting, and tautologies, i.e., equations of the form $s \simeq s$, are removed.

Note that, since the word problem is not decidable in general, a finite confluent E cannot always be obtained. In Knuth and Bendix' original procedure this could be due to *failure*³ or to non-termination of completion. For completely avoiding failure, *ordered* or *unfailing* completion was introduced [Lankford 1975, Hsiang and Rusinowitch 1987, Bachmair, Dershowitz and Plaisted 1989].

This leads to complete theorem provers for equational theories E , since for every valid equation a rewrite proof will be found after a finite number of steps of the (possibly infinite) completion procedure. Moreover, if the process terminates, it pro-

²More precisely, one rewrites the ground Skolemizations of s and t , and \succ is required to be total on such ground terms.

³Failure could occur because Knuth and Bendix considered rewriting with a terminating set of uni-directional rules, instead of ordered rewriting (applying equations in whatever direction agrees with the given reduction ordering, as explained here). Hence in their view equations had to be *oriented* into terminating rules, which fails if an equation like the commutativity axiom $f(x, y) \simeq f(y, x)$ appears.

duces a confluent system for ordered rewriting. For improving the efficiency and for reducing the number of cases of non-termination of completion, numerous additional simplification methods and *critical pair criteria* for detecting redundant inferences have been developed [Bachmair, Dershowitz and Hsiang 1986, Peterson 1990, Martin and Nipkow 1990, Bachmair 1991, Bachmair and Dershowitz 1994, Comon, Narendran, Nieuwenhuis and Rusinowitch 1998]. Indeed, nowadays completion has become the method of choice for most state-of-the-art equality reasoning systems. Since the main results for completion-based theorem proving with unit equations are particular cases of the ones given for general clauses with equality, in this chapter no further specific attention will be devoted to completion; instead, in Subsection 4.6, it will be shortly treated as an instance of saturation for general clauses.

Extending the notion of critical pair, completion procedures were developed for going beyond unit equations. For instance, for obtaining confluent sets for rewrite relations like *conditional* and *clausal* rewriting, completion procedures were designed for transforming sets of *conditional equations* (definite Horn clauses with equality, i.e., of the form $s_1 \simeq t_1 \wedge \dots \wedge s_n \simeq t_n \rightarrow s \simeq t$) [Kaplan 1984, Jouanaud and Waldmann 1986, Kounalis and Rusinowitch 1991, Ganzinger 1991], or *restricted equality* clauses [Nieuwenhuis and Orejas 1990].

The generalization of this kind of completion procedure to full first-order clauses with equality required the development of more powerful proof techniques for establishing completeness. Using the *transfinite semantic tree* method Hsiang and Rusinowitch [1991] proved the refutation completeness of *ordered paramodulation*, while Bachmair [1989] applied an extension of the so-called *proof ordering* technique for obtaining similar results.

By means of their *model generation* proof method, similar to other *forcing* techniques developed by Zhang [1988] and Pais and Peterson [1991], Bachmair and Ganzinger [1990, 1994b] proved the completeness of an inference system for full first-order clauses with equality, based on *strict superposition*: paramodulation involving only maximal (w.r.t. the ordering \succ) terms of maximal equations of clauses. Such superposition-based inference systems, as well as the model generation method, are explained in detail in Section 3 of this chapter.

1.3. Redundancy and saturation

Knuth-Bendix completion transforms sets of equations into *complete* or *saturated* ones: sets that are closed under the addition of non-joinable critical pairs, where a critical pair is joinable if it can be rewritten into a tautology $s \simeq s$.

This idea of *saturation* can be generalized: a set of formulae S is saturated for a given inference system \mathcal{I} if S is closed under \mathcal{I} , up to *redundant inferences*. Roughly, a *saturation procedure* adds conclusions of non-redundant inferences and removes *redundant formulae*. In the limit such a procedure produces a saturated set. Therefore, in the setting of first-order clauses, proving the refutation completeness of saturation amounts to showing that the empty clause \square is in S for every unsatisfiable saturated set of clauses S .

Concrete simplification methods in the context of paramodulation were discussed already in [Wos, Robinson, Carson and Shalla 1967, Slagle 1974, Loveland 1978, Peterson 1983]. Bachmair and Ganzinger [1994b] define abstract notions of redundancy for inferences and for clauses. For example, a ground clause C is redundant with respect to a set of ground clauses S if C is a logical consequence of smaller (with respect to the given clause ordering) clauses of S . These redundancy notions cover well-known practical simplification and elimination techniques, like *demodulation* (that is, simplification by rewriting with unit equations) or *subsumption* (removing a clause of the form $C\sigma \vee D$ in the presence of a more general clause C), as well as many other more powerful methods. For establishing the refutation completeness of saturation, a model is built for every saturated set not containing the empty clause. Several of the calculi and redundancy techniques explained in this chapter are available in the *Saturate* system [Nivela and Nieuwenhuis 1993, Ganzinger, Nieuwenhuis and Nivela 1999]. In Section 4 of this chapter, saturation procedures are introduced.

1.4. Computing with finite saturated sets

Due to the refined inference rules and redundancy notions, it is sometimes possible to compute a *finite* saturated set (not containing the empty clause) for a given input. In this case its satisfiability has been proved. This kind of satisfiability proving has of course many applications and is also closely related to inductive theorem proving, see [Comon and Nieuwenhuis 2000] and [Comon 2001] (Chapter 14 of this Handbook). The Spass system [Weidenbach 1997] has successfully applied saturation to prove satisfiability for all problems in the corresponding category of the 1997 CADE theorem proving competition [Sutcliffe and Suttner 1998].

Theorem proving in theories expressed by saturated sets of axioms is also interesting because more efficient proof strategies become (refutation) complete. For instance, the set-of-support strategy, which is incomplete in general for ordered inference systems and also for equality clauses, becomes complete for saturated sets S : no inferences between clauses in S are needed. Another well-known example is the completeness of rewriting with saturated sets of unit equations: saturated sets are confluent. For sets of conditional equations E or, equivalently, of Horn clauses, similar completeness results exist for *conditional rewriting* if E fulfills some syntactic requirements (e.g., in certain clauses the maximal terms must contain all variables). In general, the more such requirements are fulfilled by the saturated sets, the more restrictive proof strategies become complete. This sometimes leads to decision procedures, like the ones by rewriting for saturated sets of (conditional) equations. Computation with saturated sets is covered in Section 4.5 of this chapter. Some decision procedures are described in Section 8.2.

1.5. Paramodulation with constrained clauses

The advantages of *constrained formulae* are nowadays widely recognized in the context of logic programming. The first ideas for specific applications to

paramodulation-based theorem proving were given in [Peterson 1990, Kirchner, Kirchner and Rusinowitch 1990]. The semantics of a clause C with a constraint T , written $C \mid T$, is simply the set of all ground instances $C\sigma$ of C such that σ is a solution of T . For example, if $=$ denotes syntactic equality of terms, the constrained clause $P(x) \mid x = f(y) \wedge y > a$ denotes⁴ all ground atoms $P(f(t))$ such that t is greater than a in the given term ordering $>$. Hence if T is unsatisfiable then $C \mid T$ is a tautology.

In [Kirchner et al. 1990] ordered paramodulation inference rules were expressed for the first time by explicitly formulating the ordering and equality restrictions of the inferences by constraints at the formula level. This gives:

$$\frac{C \vee s \simeq t \mid T \quad D \mid T'}{C \vee D[t]_p \mid T \wedge T' \wedge s = D|_p \wedge OC}$$

where T and T' are the constraints inherited from the premises, the *equality constraint* $s = D|_p$ stores the unification restriction, and OC is an *ordering constraint* of the form $s > t \wedge \dots$ encoding the ordering restrictions imposed by this inference. However, the completeness results of [Kirchner et al. 1990] were limited since they required to enumerate the solutions of the constraints and *propagate* (i.e., apply) these solutions to the clause part.

Constraints are closely related to the so-called *basic strategies*, where no inferences need to be computed on subterms generated in unifiers of ancestor inference steps (like its counterpart in E -unification, called *basic narrowing* [Hullot 1980a]). It is clear that if such an inference system with inherited constraints is applied without propagation, then it is basic: the inferences only take place on the clause part C of a formula $C \mid T$, and no unifiers are ever applied to C , since the unification restrictions are simply stored in the constraint part T .

Nieuwenhuis and Rubio [1992a, 1995] showed that, in the context of superposition, indeed propagation of the equality constraints is not needed, thus proving the completeness of *basic superposition*. By using *closure substitutions*, which play the role of equality constraints, the same results were obtained independently by Bachmair and others [1992, 1995], giving additional refinements based on *term selection rules* and *redex orderings*. These developments took place independently of much earlier work in Russia by Degtyarev [1979], who used *conditional clauses* (which can in fact be seen as clauses with syntactic equality constraints) for describing a form of basic paramodulation without ordering restrictions (see also [Degtyarev and Voronkov 1986]).

In [Nieuwenhuis and Rubio 1992b] it is shown that by inheriting as well the *ordering constraints* one can restrict the search space even further without losing completeness. In [Lynch and Snyder 1993] equality, disequality and irreducibility constraints are applied for obtaining more powerful redundancy methods in basic equational completion. Finally, in [Nieuwenhuis and Rubio 1995] the use of con-

⁴Note that $>$ and $=$ are used as syntax in the constraint language. Their semantics will be a given term ordering $>$ and a given congruence (usually syntactic equality of terms) that depend on the context.

straints in theorem proving procedures is put in a more general framework based on the notion of *constraint inheritance strategies*.

The main idea in all these strategies is that the ordering and equality restrictions of the inferences can be kept in constraints and inherited between clauses. If some inference is not compatible with the required restrictions, applied to the current inference rule *and to the previous ones*, then the inference can be blocked. Therefore, for taking advantage of the constraints, algorithms for constraint satisfiability checking are required. In Section 7 of this chapter a short survey of the state of the art on such algorithms is given. Paramodulation with constrained clauses, the basic strategy and the corresponding completeness results are explained in detail in Sections 5.1 and 5.2.

1.6. Paramodulation with built-in equational theories

In principle, the aforementioned paramodulation methods apply to any set of clauses with equality, but in some cases special treatments for specific equational subsets of the axioms are preferable. On the one hand, some axioms generate many slightly different permuted versions of clauses, and for efficiency reasons it is many times better to treat all these clauses together as a single one representing the whole class. On the other hand, special treatments can avoid non-termination of completion procedures, like with $f(a, b) \simeq c$ in the presence of associativity and commutativity axioms for f . Also, some equations like the commutativity axiom are more naturally viewed as “structural” axioms (defining a congruence relation on terms) rather than as “simplifiers” (defining a reduction relation). This allows one to extend completion procedures in order to deal with congruence classes of terms instead of single terms, i.e., working with a *built-in* equational theory E , and performing rewriting and completion with special E -matching and E -unification algorithms.

Early results on paramodulation and rewriting *modulo* E were given by Plotkin [1972], Slagle [1974] and Lankford and Ballantine [1977] and *extended E -rewriting* was defined by Peterson and Stickel [1981]. Several E -completion procedures for the equational case were developed e.g. in [Lankford and Ballantyne 1977, Huet 1980, Peterson and Stickel 1981, Jouannaud 1983, Jouannaud and Kirchner 1986, Bachmair and Dershowitz 1989]. Special attention has always been devoted to the case where E includes axioms of associativity and commutativity (AC), which occur very frequently in practical applications, and are well-suited for being built in due to their permutative nature.

The generalization of these E -completion techniques to full first-order clauses with equality has been studied in e.g. [Paul 1992, Wertz 1992, Rusinowitch and Vigneron 1995, Bachmair and Ganzinger 1994a], usually with particular treatments for the AC case. Paramodulation *modulo* E then becomes roughly the following rule, which has one conclusion for each σ in $U_E(s, D|_p)$, a *minimal complete* set of E -unifiers of $D|_p$ and s :

$$\frac{C \vee s \simeq t \quad D}{(C \vee D[t]_p)\sigma} \quad \text{for all unifiers } \sigma \text{ in } U_E(s, D|_p)$$

Note that in general there is no unique most general E -unifier for a given E -unification problem, and that new variables may appear: for example, if f is an AC-symbol, then $f(x, a)$ and $f(y, b)$ have the two AC-unifiers $\sigma_1 = \{x \mapsto b, y \mapsto a\}$ and $\sigma_2 = \{x \mapsto f(b, z), y \mapsto f(a, z)\}$. In Section 6 of this chapter we introduce some of the main techniques on paramodulation modulo equational theories.

1.7. Basic paramodulation with built-in equational theories

For an equational theory E , the number of E -unifiers of two terms may be large. For instance, the cardinality of a minimal complete set of AC-unifiers is doubly exponential in general [Domenjoud 1992] (in a sense, this is also an upper bound [Kapur and Narendran 1992]). Hence a single E -paramodulation inference can generate a large number of new clauses.

Therefore, equality constraints become extremely useful in this context. In constrained E -paramodulation, instead of E -unifying the terms, the unification problem is stored in the constraint. Hence in the constrained superposition inference rule given in Section 1.5, the semantics of the symbol ‘=’ in the equality constraint $s = D|_p$ becomes E -equality. Dealing with a constrained clause $C \mid s = t$ can be much more efficient than having n clauses C_1, \dots, C_n , one for each E -unifier of s and t , since many inferences are computed at once, and each inference generates one single conclusion. Furthermore, computing E -unifiers is not needed. A clause C with an E -equality constraint T can be proved redundant by means of efficient (sound, but possibly incomplete) methods for detecting unsatisfiable T . If C is the empty clause, a contradiction has been derived if, and only if, the constraint part T is satisfiable, and hence in this case refutation completeness requires a semi-decision procedure for detecting these contradictions. Such a procedure exists for every finite E .

The completeness of such a fully basic strategy for the AC-case (combined with ordering constraints) was first proved in [Nieuwenhuis and Rubio 1994, Nieuwenhuis and Rubio 1997], although the first results on (almost basic) constrained deduction methods modulo AC were reported in [Vigneron 1994]. The basicness restriction is considered to “have been a key strategy” by McCune [1997b] in his celebrated AC-paramodulation-based proof of the Robbins problem. In Section 6.3 of this chapter basic paramodulation modulo AC is explained.

2. Preliminaries

In order to keep this chapter self-contained, here we introduce the main basic tools used: terms, rewriting, term orderings, first-order equality clauses and equality

Herbrand interpretations. Most (if not all) of our definitions are consistent with [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook).

2.1. Terms and (rewrite) relations

Let \mathcal{F} be a *signature*, a (finite) set of function symbols with an arity function $\text{arity}: \mathcal{F} \rightarrow \mathbb{N}$ and let \mathcal{X} be a set of variable symbols. Function symbols f with $\text{arity}(f) = n$ are called *n-ary* symbols (when $n = 1$, one says *unary* and when $n = 2$, *binary*). If $\text{arity}(f) = 0$, then f is a *constant symbol*. The set of *first-order terms* over \mathcal{F} and \mathcal{X} , denoted by $\mathcal{T}(\mathcal{F}, \mathcal{X})$, is the smallest set containing \mathcal{X} such that $f(t_1, \dots, t_n)$ is in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ whenever $f \in \mathcal{F}$, $\text{arity}(f) = n$, and $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Similarly, $\mathcal{T}(\mathcal{F})$ is the set of variable free or *ground* terms. Note that $\mathcal{T}(\mathcal{F}) = \emptyset$ if there are no constant symbols in \mathcal{F} . As usual, along this chapter it is therefore assumed that there is at least one constant symbol in \mathcal{F} .

A *position* is a sequence of positive integers. If p is a position and t is a term, then by $t|_p$ we denote the *subterm of t at position p* : we have $t|_\lambda = t$ (where λ denotes the empty sequence) and $f(t_1, \dots, t_n)|_{i.p} = t_i|_p$ if $1 \leq i \leq n$ (and is undefined if $i > n$). We also write $t[s]_p$ to denote the term obtained by replacing in t the subterm at position p by the term s . For example, if t is $f(a, g(b, h(c)), d)$, then $t|_{2.2.1} = c$, and $t[d]_{2.2} = f(a, g(b, d), d)$. We say that a variable (or function symbol) x *occurs* (at position p) in a term t if $t|_p$ is (rooted by) x . By $\text{vars}(t)$ we denote the set of all variables occurring in t . If t is a term of the form $f(t_1, \dots, t_n)$, then we define $\text{top}(t)$ to be the function symbol f . The syntactic equality of two terms s and t will be denoted by $s \equiv t$.

A *substitution* σ is a mapping from variables to terms. It can be extended to a function from terms to terms in the usual way: using a postfix notation, $t\sigma$ denotes the result of simultaneously replacing in t every $x \in \text{Dom}(\sigma)$ by $x\sigma$. Here substitutions are sometimes written as sets of pairs $x \mapsto t$, where x is a variable and t is a term. For example, if σ is $\{x \mapsto f(b, y), y \mapsto a\}$, then $g(x, y)\sigma$ is $g(f(b, y), a)$ (this example illustrates the *simultaneous* replacement: applying σ “from left to right” yields $g(f(b, a), a)$, which is not the intended meaning).

A substitution σ is *ground* if its range is $\mathcal{T}(\mathcal{F})$. Unless stated otherwise, we will assume that ground substitutions σ applied to a term t are also *grounding*, that is, $\text{vars}(t) \subseteq \text{Dom}(\sigma)$, and hence $t\sigma$ is ground. A term t *matches* a term s if $s\sigma \equiv t$ for some σ . Then t is called an *instance* of s .

A term t is *unifiable* with a term s if $s\sigma \equiv t\sigma$ for some substitution σ . Then σ is called a *unifier* of s and t . Furthermore, a substitution σ is called a *most general unifier* of s and t , denoted $\text{mgu}(s, t)$, if $s\sigma \equiv t\sigma$, and for every other unifier θ of s and t , it holds that $s\theta \equiv s\sigma\sigma' \equiv t\theta \equiv t\sigma\sigma'$ for some σ' , that is, roughly, if every other unifier θ is a particular instance of σ . We sometimes speak about *the mgu* of s and t because it is unique up to variable renaming, see [Baader and Snyder 2001] (Chapter 8 of this Handbook) for details and for unification algorithms computing *mgu*'s.

A *multiset* over a set S is a function $M: S \rightarrow \mathbb{N}$. The union and intersection of

multisets are defined as usual by $M_1 \cup M_2(x) = M_1(x) + M_2(x)$, and $M_1 \cap M_2(x) = \min(M_1(x), M_2(x))$. We also use a set-like notation: $M = \{a, a, b\}$ denotes $M(a) = 2$, $M(b) = 1$, and $M(x) = 0$ for $x \neq a$ and $x \neq b$. A multiset M is *empty* if $M(x) = 0$ for all $x \in S$.

If \rightarrow is a binary relation, then \leftarrow is its inverse, \leftrightarrow is its symmetric closure, \rightarrow^+ is its transitive closure and \rightarrow^* is its reflexive-transitive closure. We write $s \rightarrow^1 t$ if $s \rightarrow^* t$ and there is no t' such that $t \rightarrow t'$. Then t is called *irreducible* and a *normal form* of s (w.r.t. \rightarrow). The relation \rightarrow is *well-founded* or *terminating* if there exists no infinite sequence $s_1 \rightarrow s_2 \rightarrow \dots$ and it is *confluent* or *Church-Rosser* if the relation $\leftarrow^* \circ \rightarrow^*$ is contained in $\rightarrow^* \circ \leftarrow^*$. It is *locally confluent* if $\leftarrow \circ \rightarrow \subseteq \rightarrow^* \circ \leftarrow^*$. By Newman's lemma, terminating locally-confluent relations are confluent. A relation \rightarrow on terms is *monotonic* if $s \rightarrow t$ implies $u[s]_p \rightarrow u[t]_p$ for all terms s, t and u and positions p . A *congruence* is a reflexive, symmetric, transitive and monotonic relation on terms.

An *equation* is a multiset $\{s, t\}$, denoted $s \simeq t$ or, equivalently, $t \simeq s$. A *rewrite rule* is an ordered pair (s, t) , written $s \Rightarrow t$, and a set of rewrite rules R is a *rewrite system*. The rewrite relation with R on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, denoted \rightarrow_R , is the smallest monotonic relation such that $l\sigma \rightarrow_R r\sigma$ for all $l \rightarrow r \in R$ and all σ . If $s \rightarrow_R t$ then we say that s *rewrites into* t with R . We say that R is terminating, confluent, etc. if \rightarrow_R is. A rewrite system R is called *convergent* if it is confluent and terminating. It is not difficult to see that then every term t has a unique normal form w.r.t. \rightarrow_R , denoted by $nf_R(t)$, and $s = t$ is a logical consequence of R (where R is seen as a set of equations) if and only if $nf_R(s) = nf_R(t)$. Sometimes the congruence relations (on $\mathcal{T}(\mathcal{F})$) \leftrightarrow_R^* (or \leftrightarrow_E^*) are denoted by R^* (E^*) or $=_R$ ($=_E$).

2.2. Term orderings

A (strict partial) ordering \succ is a transitive and irreflexive binary relation. An ordering \succ on terms is *stable* (or *closed*) under substitutions if $s \succ t$ implies $s\sigma \succ t\sigma$ for all s, t and σ ; it fulfills the *subterm property* if $u[s]_p \succ s$ for all s, u and $p \neq \lambda$. It is *total* on $\mathcal{T}(\mathcal{F})$ if for all s and t in $\mathcal{T}(\mathcal{F})$, either $s = t$ or $s \succ t$ or $t \succ s$; if $=$ is a congruence different from syntactic equality, we speak about totality *up to* $=$.

A *rewrite ordering* is a monotonic ordering stable under substitutions; a *reduction ordering* is a well-founded rewrite ordering, and a *simplification ordering* is a rewrite ordering with the subterm property.

The following properties are not difficult to check: a reduction ordering total on $\mathcal{T}(\mathcal{F})$ is necessarily a simplification ordering on $\mathcal{T}(\mathcal{F})$; by Kruskal's theorem, simplification orderings are well-founded (for finite, fixed-arity signatures); and a rewrite system R is terminating if and only if all its rules are contained in a reduction ordering \succ , i.e., $l \succ r$ for every $l \rightarrow r \in R$ (in fact, then \rightarrow_R^+ is itself a reduction ordering).

Let \succ be an ordering on terms and let $=$ be a congruence relation. Then \succ is called *compatible* with $=$ if $s' = s \succ t = t'$ implies $s' \succ t'$ for all s, s', t and t' . If E is a set of equations, then \succ is called *E-compatible* if it is compatible with $=_E$. Note

that if \succ is E-compatible, $s =_E t$ implies $s \not\succ t$ and $t \not\succ s$.

Let \succ be an ordering on terms and let $=$ be a congruence relation such that \succ is compatible with $=$. Then these relations induce relations on tuples and multisets of terms as follows.

The *lexicographic (left to right) extension of \succ with respect to $=$* is the relation \succ^{lex} on n -tuples of terms defined by:

$$\langle s_1, \dots, s_n \rangle \succ^{lex} \langle t_1, \dots, t_n \rangle \quad \text{if} \quad s_1 = t_1, \dots, s_{k-1} = t_{k-1} \text{ and } s_k \succ t_k$$

for some k in $1 \dots n$. It is well-known that, if \succ is well founded, so is \succ^{lex} .

The *multiset extension of $=$* is defined as the smallest relation $==$ on multisets of terms such that $\emptyset == \emptyset$ and

$$S \cup \{s\} == S' \cup \{t\} \text{ if } s = t \wedge S == S'$$

The *multiset extension of \succ with respect to $=$* is defined as the smallest ordering \succ (or \succ_{mul}) on multisets of terms such that

$$M \cup \{s\} \succ N \cup \{t_1, \dots, t_n\} \text{ if } M == N \text{ and } s \succ t_i \text{ for all } i \in 1 \dots n$$

Sometimes the notation \succ is used without explicitly indicating which is the congruence $=$. In these cases $=$ is assumed to be the syntactic equality relation \equiv on terms. If \succ is well founded on S , so is \succ on finite multisets over S [Dershowitz and Manna 1979].

A way to define suitable orderings for practical purposes (like termination proving or automated deduction) is to construct them directly from a well-founded *precedence*, an ordering \succ_F on \mathcal{F} . This is done in the so-called *path orderings*, like the *lexicographic path ordering* (LPO) or the *recursive path ordering (with status)* (RPO) [Kamin and Levy 1980, Dershowitz 1982].

Let \succ_F be a precedence and let \mathcal{F} be the disjoint union of two sets *lex* and *mul*, the symbols with lexicographic and multiset *status*, respectively. By $=_{mul}$ we denote the equality of ground terms up to the permutation of direct arguments of symbols with multiset status: $f(s_1, \dots, s_m) =_{mul} g(t_1, \dots, t_n)$ if $f = g$ and hence $m = n$, and $s_{\pi(i)} =_{mul} t_i$ for $1 \leq i \leq n$ and where π is a permutation of $1 \dots n$ which is the identity if $f \in lex$.

In this setting, RPO is defined as follows: $s \succ_{rpo} x$ if x is a variable that is a proper subterm of s or else $s \equiv f(s_1 \dots s_n) \succ_{rpo} t \equiv g(t_1 \dots t_m)$ if at least one of the following conditions holds:

- $s_i \succ_{rpo} t$ or $s_i =_{mul} t$, for some $i \in \{1 \dots n\}$
- $f \succ_F g$, and $s \succ_{rpo} t_j$, for all j in $\{1 \dots m\}$
- $f \equiv g$ (and hence $n=m$) and $f \in mul$ and $\{s_1, \dots, s_n\} \succ_{rpo} \{t_1, \dots, t_n\}$
- $f \equiv g$ (and hence $n=m$) and $f \in lex$, $\langle s_1, \dots, s_n \rangle \succ_{rpo}^{lex} \langle t_1, \dots, t_n \rangle$, and $s \succ_{rpo} t_j$, for all j in $\{1 \dots n\}$

where \succ_{rpo}^{lex} and \succ_{rpo} are, respectively, the lexicographic and multiset extensions of \succ_{rpo} with respect to $=_{mul}$.

The *lexicographic path ordering* (LPO) is defined as the particular case of an RPO where $\mathcal{F} = \text{lex}$, i.e., where all symbols have a lexicographic status.

It is known that RPO is a reduction ordering on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, which is moreover total on $\mathcal{T}(\mathcal{F})$ up to $=_{mul}$ (and hence in case of LPO, total up to \equiv) if $\succ_{\mathcal{F}}$ is total on \mathcal{F} [Kamin and Levy 1980, Dershowitz 1982].

LPO's are useful for extending reduction orderings \succ that are total up to a congruence $=$ (like RPO is total up to $=_{mul}$), to reduction orderings total up to \equiv . This extension is obtained by a lexicographic combination \succ_t whose first component is \succ , and whose second component is a total LPO \succ_{lpo} , that is, $s \succ_t t$ if either $s \succ t$ or $s = t$ and $s \succ_{lpo} t$.

It is not difficult to see that RPO is C-compatible (C for commutativity) if commutative symbols have multiset status, but it is not AC-compatible.

2.3. Equality clauses and Herbrand interpretations

A clause is a pair of finite multisets of equations Γ (the *antecedent*) and Δ (the *succedent*), denoted by $\Gamma \rightarrow \Delta$. We sometimes use a comma in clauses to denote the union of multisets or the inclusion of equations in multisets; for example, we write $s \simeq t, \Gamma, \Gamma' \rightarrow \Delta$ instead of $\{s \simeq t\} \cup \Gamma \cup \Gamma' \rightarrow \Delta$. Clauses $e_1, \dots, e_n \rightarrow e'_1, \dots, e'_m$ are sometimes (equivalently) written as a disjunction of equations and negated equations $\neg e_1 \vee \dots \vee \neg e_n \vee e'_1 \vee \dots \vee e'_m$. Hence, the e_i are called the *negative* equations, and the e'_j the *positive* equations, respectively, of the clause.

A clause $\Gamma \rightarrow \Delta$ is called a *Horn clause* if Δ contains at most one equation. The *empty clause* \square is a clause $\Gamma \rightarrow \Delta$ where both Γ and Δ are empty. A *positive* (resp. *negative*) clause is a clause $\Gamma \rightarrow \Delta$ where Γ (resp. Δ) is empty, and a *unit* clause is a clause with exactly one literal.

We will use all aforementioned notions and notations defined for terms t , like $t|_p$, $t[s]_p$, $\text{vars}(t)$, $t\sigma$, etc., as well for equations and clauses in the expected way. For example, a term u occurs in a clause $\Gamma \rightarrow \Delta$ if $t \simeq s \in \Gamma \cup \Delta$ and $t|_p \simeq u$ for some position p .

Let R be a set of ground equations (or rewrite rules). Then the congruence \leftrightarrow_R^* defines an equality Herbrand interpretation I : the domain of I is $\mathcal{T}(\mathcal{F})$, each n -ary function symbol f of \mathcal{F} is interpreted as the function f_I where $f_I(t_1, \dots, t_n)$ is the term $f(t_1, \dots, t_n)$, and where the only predicate \simeq is interpreted by $s \simeq t$ if $s \leftrightarrow_R^* t$. The interpretation I defined by R in this way will be denoted by R^* . We write $s = t \in I$ if $s \leftrightarrow_R^* t$. I satisfies (is a model of) a ground clause $\Gamma \rightarrow \Delta$, denoted $I \models \Gamma \rightarrow \Delta$, if $I \not\models \Gamma$ or $I \cap \Delta \neq \emptyset$. The empty clause \square is hence satisfied by no interpretation. I satisfies a non-ground clause C if I satisfies all ground instances of C . I satisfies a set of clauses S , denoted by $I \models S$, if it satisfies every clause in S . A clause C is a logical consequence of (or C follows from) a set of clauses S , denoted by $S \models C$, if C is satisfied by every model of S .

2.4. Constraints and constrained clauses

An (*ordering and equality*) *constraint* is a quantifier-free first-order formula built over the binary predicate symbols $>$ and $=$ relating terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Regarding semantics, the constraints are interpreted in $\mathcal{T}(\mathcal{F})$, and $=$ is interpreted as some congruence $=_c$ on $\mathcal{T}(\mathcal{F})$ (like syntactic equality or AC-equality) and $>$ is interpreted as a given reduction ordering \succ on ground terms that is total up to $=_c$. Hence a *solution* of a constraint T is a ground substitution σ with domain $\text{vars}(T)$ and such that $T\sigma$ evaluates to true for the given $=_c$ and \succ . If a solution for T exists, then T is called *satisfiable*. If every ground substitution with domain $\text{vars}(T)$ is a solution of T then T is a *tautology*.

A *constrained clause* is a pair $C \mid T$ where C is a clause and T is a constraint. A *ground instance* of $C \mid T$ is a ground clause $C\sigma$ where σ is a solution of T . The semantics of $C \mid T$ is the set of all its ground instances. Hence, by definition, an interpretation I satisfies $C \mid T$ if $I \models C\sigma$ for every ground instance $C\sigma$ of $C \mid T$. Therefore, clauses with unsatisfiable constraints are tautologies. A clause $C \mid T$ is the *constrained empty clause*, denoted as well by \square , if C is empty and T is satisfiable. Constrained clauses $C \mid T$ where T is a tautology are sometimes denoted by C , omitting the constraint part T .

3. Paramodulation calculi

A logical *inference* is a step by which from a multiset of zero or more constrained clauses (the *premises*) a new constrained clause (the *conclusion*) is obtained. An *inference rule* \mathcal{R}

$$\frac{C_1 \mid T_1 \dots C_n \mid T_n}{D \mid T} \quad \text{if } \text{condition}$$

is (a finite representation of) the set of inferences where from the multiset of clauses of the form $\{C_1 \mid T_1 \dots C_n \mid T_n\}$ one can infer $D \mid T$ if *condition* holds. One such an inference is called an *inference by* \mathcal{R} . An *inference system* \mathcal{I} is a set of inference rules. An *inference by* \mathcal{I} is an inference by one of the rules of \mathcal{I} . We will frequently consider inference rules where premises or conclusions have constraints that are tautologies and hence these constraints are omitted.

An inference rule \mathcal{R} is *correct* if, for all inferences by \mathcal{R} , the conclusion is a logical consequence of the premises, and an inference system is correct if all its rules are correct. A set of constrained clauses S is *closed* under \mathcal{I} if for every inference by \mathcal{I} with premises in S , the corresponding conclusion is in S . \mathcal{I} is *refutation complete* if $\square \in S$ for every unsatisfiable set of constrained clauses S closed under \mathcal{I} . All inference systems in the remainder of this chapter are easily proved correct, and we will focus on completeness.

3.1. The model generation method

We start with a simple example on ground Horn clauses in order to introduce the *model generation* method, the standard technique for establishing the completeness of ordered paramodulation calculi that will be used throughout this chapter. Note that if $C \mid T$ is a constrained clause where C is ground and T is satisfiable, then it is equivalent to $C \mid \top$ where \top denotes a tautological constraint. Hence in the remainder of this section constraints will be omitted.

In the following, let \succ be a given total reduction ordering on $\mathcal{T}(\mathcal{F})$, and let $s \succeq t$ denote $s \succ t \vee s \equiv t$. The inference system \mathcal{G} for ground Horn clauses with equality is the following:

superposition right:

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t} \quad \text{if} \quad \begin{array}{l} s|_p \equiv l, l \succ r, s \succ t, \text{ and} \\ l \succ u \text{ for all } u \text{ occurring in } \Gamma', \text{ and} \\ s \succ v \text{ for all } v \text{ occurring in } \Gamma \end{array}$$

superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma, s \simeq t \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta} \quad \text{if} \quad \begin{array}{l} s|_p \equiv l, l \succ r, s \succ t, \text{ and} \\ l \succ u \text{ for all } u \text{ occurring in } \Gamma', \text{ and} \\ s \succeq v \text{ for all } v \text{ occurring in } \Gamma, \Delta \end{array}$$

equality resolution:

$$\frac{\Gamma, s \simeq s \rightarrow \Delta}{\Gamma \rightarrow \Delta} \quad \text{if} \quad s \succeq v \text{ for all } v \text{ occurring in } \Gamma, \Delta$$

Let us remark that the equality resolution rule is named after the fact that it encodes a resolution inference with the reflexivity axiom of equality $x \simeq x$,

It is sometimes said that in the superposition rules the inferences take place *with* the term l *on* the term s , and that the inference *involves* s and l . Note that in \mathcal{G} , superposition right inferences involve only terms s and l that are *strictly maximal* in their respective premises, that is, they are bigger w.r.t. \succ than all other occurrences of terms in these premises. Superposition left takes place also with strictly maximal terms, but on (possibly non-strictly) *maximal* terms (that is, they are larger than or equal to all terms in their premise).

In order to prove the refutation completeness of \mathcal{G} we first define the following total ordering \succ_c on ground clauses. If C is a clause

$$s_1 = s'_1, \dots, s_n = s'_n \rightarrow t_1 = t'_1, \dots, t_m = t'_m$$

then we define $ms(C)$ as the multiset:

$$\{\{s_1, s_1, s'_1, s'_1\}, \dots, \{s_n, s_n, s'_n, s'_n\}, \{t_1, t'_1\}, \dots, \{t_m, t'_m\}\}$$

Finally, let \succ_c be the ordering on clauses defined by comparing these expressions by the two-fold multiset extension of \succ , that is, $C \succ_c D$ if $ms(C)(\succ_{mul})_{mul}ms(D)$. The result is a total ordering on ground clauses⁵.

Now we come to the key to the model generation method. Our aim is to prove the completeness of \mathcal{G} . We do this by showing that, if S is a set of ground Horn clauses closed under \mathcal{G} and $\Box \notin S$, then S is satisfiable. The satisfiability proof of S is of a constructive nature: first, an equality Herbrand interpretation will be built, and second, it will be shown that this interpretation is a model of S .

We now informally explain the first part. The interpretation we build will be the congruence R^* induced by a set of ground rewrite rules R , where each rule in R has been *generated* by some clause of S (hence the name “model generation”). The generation process of R is defined by induction on \succ_c . Each clause C in S generates a rule or not, depending on the set R_C of rules generated by clauses D of S with $C \succ_c D$ (and on the congruence R_C^* induced by R_C). These ideas are formalised as follows:

3.1. DEFINITION. (Model generation) Let C be a clause in S . Then $Gen(C) = \{l \Rightarrow r\}$, and C is said to *generate* the rule $l \Rightarrow r$, if, and only if, C is of the form $\Gamma \rightarrow l \simeq r$ and the three following conditions hold:

1. $R_C^* \not\models C$,
2. $l \succ r$ and $l \succ u$ for all u occurring in Γ
3. l is irreducible by R_C

where $R_C = \bigcup_{C \succ_c D} Gen(D)$. In all other cases $Gen(C) = \emptyset$. Finally, R denotes the set of all rules generated by clauses of S , that is, $R = \bigcup_{D \in S} Gen(D)$.

Let us analyse the three conditions. The first one states that a clause only contributes to the model if it does not hold in the partial model built so far and hence we are *forced* to extend this partial model. The second one states that a clause can only generate a rule $l \Rightarrow r$ if l is the strictly maximal term of the clause. The third condition, stating that l is irreducible by the rules generated so far, is, together with the second one, the key for showing that R is convergent, from which the completeness result quite easily follows:

3.2. LEMMA. *For every set of ground clauses S , the set of rules R generated for S is convergent (i.e., confluent and terminating). Furthermore, if $R_C^* \models C$ then $R^* \models C$ for all ground C .*

PROOF. Evidently, R is terminating since $l \succ r$ for all its rules $l \Rightarrow r$. To prove confluence, it suffices to show *local* confluence, which in the ground case is well-known (and easily shown) to hold if there are no two different rules $l \Rightarrow r$ and $l' \Rightarrow r'$ where l' is a subterm of l . This property is fulfilled: clearly when a clause

⁵Roughly, \succ_c compares the multisets of all equations occurring in the clauses, but where in addition terms occurring negatively have slightly more weight than the ones occurring positively; in fact, in order to make \succ_c total on ground clauses, the information of which equations are positive and which ones are negative has to be present anyway.

C in S generates $l \Rightarrow r$, no such $l' \Rightarrow r'$ is in R_C ; but if $l' \Rightarrow r'$ is generated by a clause D with $D \succ_c C$ then, by definition of \succ_c , we must have $l' \succ l$ and hence l' cannot be a subterm of l either.

To show $R_C^* \models C$ implies $R^* \models C$, let C be $\Gamma \rightarrow \Delta$, and assume $R_C^* \models C$. If $R_C^* \models \Delta$ then $R^* \models \Delta$ since $R \supseteq R_C$. Otherwise, $R_C^* \not\models \Gamma$. Then $R^* \not\models \Gamma$ follows from the fact a term t occurring negatively in a clause is bigger than the same t occurring positively: all rules in $R \setminus R_C$ are generated by clauses bigger than C , and hence have left hand sides that are too big to reduce any term occurring in Γ . Since R is convergent this implies $R^* \not\models \Gamma$. \square

3.3. THEOREM. *The inference system \mathcal{G} is refutation complete for ground Horn clauses.*

PROOF. Let S be a set of ground Horn clauses that is closed under \mathcal{G} and such that $\square \notin S$. We prove that then S is satisfiable by showing that R^* is a model for S . We proceed by induction on \succ_c , that is, we derive a contradiction from the existence of a minimal (w.r.t. \succ_c) clause C in S such that $R^* \not\models C$. There are a number of cases to be considered, depending on the occurrences in C of its maximal term s , i.e., the term s such that $s \succeq u$ for all terms u in C (s is unique since \succ is total on $\mathcal{T}(\mathcal{F})$):

1. s occurs only in the succedent and C is $\Gamma \rightarrow s \simeq s$. This is not possible since $R^* \not\models C$.
2. s occurs only in the succedent and C is $\Gamma \rightarrow s \simeq t$ with $s \neq t$. Since $R^* \not\models C$, we have $R^* \supseteq \Gamma$ and $s \simeq t \notin R^*$, i.e., C has not generated the rule $s \Rightarrow t$. This must be because s is reducible by some rule $l \Rightarrow r \in R_C$. Assume $l \Rightarrow r$ has been generated by a clause C' of the form $\Gamma' \rightarrow l \simeq r$. Then there exists an inference by superposition right:

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t}$$

whose conclusion D has only terms u with $s \succ u$, and hence $C \succ_c D$. Moreover, D is in S and $R^* \not\models D$, since $R^* \supseteq \Gamma \cup \Gamma'$ and $s[r]_p \simeq t \notin R^*$ (since otherwise $s[l]_p \simeq t \in R^*$). This contradicts the minimality of C .

3. s occurs in the antecedent and C is $\Gamma, s \simeq s \rightarrow \Delta$. Then there exists an inference by equality resolution:

$$\frac{\Gamma, s \simeq s \rightarrow \Delta}{\Gamma \rightarrow \Delta}$$

for whose conclusion D it holds that $C \succ_c D$. Moreover, D is in S and $R^* \not\models D$, which is a contradiction as in the previous case.

4. s occurs in the antecedent and C is $\Gamma, s \simeq t \rightarrow \Delta$ with $s \succ t$. Since $R^* \not\models C$, we have $s \simeq t \in R^*$ and since R is convergent, s and t must have the same normal forms w.r.t. R , so s must be reducible by some rule $l \Rightarrow r \in R$. Assume $l \Rightarrow r$

has been generated by a clause C' of the form $\Gamma' \rightarrow l \simeq r$. Then there exists an inference by superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma, s[l]_p \simeq t \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta}$$

for whose conclusion D it holds that $C \succ_c D$. Moreover, D is in S and $R^* \not\models D$, which again contradicts the minimality of C . \square

The following example shows how the rewrite system R changes during a closure of a set of ground clauses and that, although for the intermediate sets the obtained R^* is not a model, the R^* obtained for the closed set is a model.

3.4. EXAMPLE. Consider the lexicographic path ordering generated by the precedence $f \succ_{\mathcal{F}} a \succ_{\mathcal{F}} b \succ_{\mathcal{F}} c \succ_{\mathcal{F}} d$. The following table shows in the left column the ground Horn clauses (sorted with respect to the ordering) at each closure step, in which the first one is the initial set, and in the right column the set R corresponding to each intermediate set. The maximal term of every clause is underlined and the subterms of the clauses involved in the inference are framed.

S	R
$\rightarrow \underline{c} \simeq d$ $\underline{f(d)} \simeq d \rightarrow a \simeq b$ $\rightarrow \underline{f(\underline{c})} \simeq d$	$c \Rightarrow d$
$\rightarrow \underline{c} \simeq d$ $\rightarrow \underline{f(d)} \simeq d$ $\underline{f(d)} \simeq d \rightarrow a \simeq b$ $\rightarrow \underline{f(c)} \simeq d$	$c \Rightarrow d$ $f(d) \Rightarrow d$
$\rightarrow \underline{c} \simeq d$ $d \simeq d \rightarrow \underline{a} \simeq b$ $\rightarrow \underline{f(d)} \simeq d$ $\underline{f(d)} \simeq d \rightarrow a \simeq b$ $\rightarrow \underline{f(c)} \simeq d$	$c \Rightarrow d$ $a \Rightarrow b$ $f(d) \Rightarrow d$

Let us conclude this section with a remark on additional ordering restrictions. In superposition left as well as in equality resolution, it is possible to strengthen the conditions in such a way that only one negative literal becomes eligible for inferences. For example, in superposition left on an equation $s \simeq t$, one can require that $t \succ t'$ for all equations $s \simeq t'$ in Γ , that is, we use the maximal equation rather than just the maximal term; if two equations have the same maximal terms, we compare the other terms. Similarly, in equality resolution we can require $s \succ t'$ for all equations $s \simeq t'$ in Γ . In the inference system for general clauses (see Subsection 3.5) we have included these restrictions, since such comparisons between equations are needed there anyway. We did not consider them for \mathcal{G} for simplicity reasons, and also because by means of *selection of negative equations* we will be able to obtain stronger results in a simpler way (see Subsection 3.6).

3.2. Non-equality predicates

In this framework, equality can be considered to be the only predicate, since for every other predicate symbol p , (positive or negative) atoms $p(t_1 \dots t_n)$ can be expressed as (positive or negative) equations $p(t_1 \dots t_n) \simeq \text{true}$, where true is a new special symbol, and where p is considered as a function symbol rather than as a predicate symbol. Note however that, in order to avoid meaningless expressions in which predicate symbols occur at proper subterms one should adopt a two-sorted type discipline on terms in the encoding.

It is easy to see that this transformation preserves satisfiability. Very roughly: one can “translate” the interpretations such that a ground atom is true in a Herbrand interpretation I if and only if in the equality Herbrand interpretation I' over the modified signature the term $p(t_1 \dots t_n)$ is congruent to true . Be we remark that I and I' are not isomorphic since two ground atoms that are false in I need not be in the same congruence class of I' .

After this satisfiability preserving transformation, ordered resolution (ground) inferences of the form:

$$\frac{\Gamma' \rightarrow A \quad \Gamma, A \rightarrow \Delta}{\Gamma', \Gamma \rightarrow \Delta} \quad \text{if} \quad A \succ \Gamma' \text{ and } A \succeq \Gamma, \Delta.$$

become a special case of superposition left:

$$\frac{\Gamma' \rightarrow A \simeq \text{true} \quad \Gamma, A \simeq \text{true} \rightarrow \Delta}{\Gamma', \Gamma, \text{true} \simeq \text{true} \rightarrow \Delta}$$

combined with equality resolution (or simplification, as we will see) for eventually eliminating the trivial equation $\text{true} \simeq \text{true}$.

For efficiency reasons it is convenient to make true small in the ordering. Sometimes it is also useful to take into account that p is a predicate symbol when handling the ordering restrictions. For example, in orderings like RPO, if the predicate symbols p are bigger in the precedence than function symbols then $p \succ_{\mathcal{F}} q$ implies $p(t_1, \dots, t_n)\sigma \succ_{lpo} q(s_1, \dots, s_m)\sigma$ for all ground σ .

3.3. Clauses with variables

Up to now, in this section we have only dealt with ground clauses. If we consider that a non-ground clause represents the set of all its ground instances⁶, a refutation complete method for the non-ground case would be to systematically enumerate all ground instances of the clauses, and to perform inferences by \mathcal{G} between those instances. But fortunately it is possible to perform inferences between non-ground clauses, covering in one step a possibly large number of ground inferences. We now adapt \mathcal{G} according to this view.

For example, at the ground level, in the superposition right inference

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t} \quad \text{if} \quad \begin{array}{l} s|_p \equiv l, l \succ r, s \succ t, \text{ and} \\ l \succ u \text{ for all } u \text{ occurring in } \Gamma', \text{ and} \\ s \succ v \text{ for all } v \text{ occurring in } \Gamma \end{array}$$

we required $s|_p$ and l to be the same term. At the non-ground level, this becomes a constraint $s|_p = l$ on the possible instances of the conclusion, that is, the conclusion is a constrained clause $D \mid T$. Hence if the conclusion is $D \mid s|_p = l \wedge \dots$, the instances $D\sigma$ for which $s|_p\sigma \neq l\sigma$ are not created. The same is done for the ordering restrictions. For instance, instead of requiring $l \succ r$ as a condition of the inference, it becomes part of the constraint of the conclusion, excluding those instances $D\sigma$ of the conclusion that correspond to ground inferences between instances of the premises for which $l\sigma \succ r\sigma$ does not hold:

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t \mid s|_p = l \wedge l \succ r \wedge s \succ t \wedge \dots}$$

Note that here we have written the inference rule without constraints in its premises, since at this point we are only interested in the constraints that are generated in this concrete inference. In Section 5 paramodulation with constraints inherited from the premises will be considered in detail.

This inference rule can be further restricted with the additional condition stating that the inference is not necessary if $s|_p$ is a variable. This shows that, by working on the non-ground level, certain inferences between ground instances of the premises turn out to be redundant: at the non-ground level we do not perform, for an instance with σ , the inferences *inside* σ (also called inferences *below variables*), that is, on positions $s\sigma|_p$ where $s|_{p'}$ is a variable for some prefix p' of p .

Note that, as usual, it may be necessary to rename variables in the premises in order to avoid name clashes: the premises C and D are assumed to fulfill $\text{vars}(C) \cap \text{vars}(D) = \emptyset$.

Now we define the inference system \mathcal{H} for non-ground Horn clauses, writing $s \succ \Gamma$ as a shorthand for the constraint $s \succ u_1 \wedge s \succ v_1 \wedge \dots \wedge s \succ u_n \wedge s \succ v_n$ if

⁶By Herbrand's theorem, considering only the ground instances preserves satisfiability; in fact, this is a consequence of (the proof of) Theorem 3.10.

Γ is a multiset of equations $\{u_1 \simeq v_1, \dots, u_n \simeq v_n\}$ (and similarly, we write $s \geq \Gamma$ for $s \geq u_1 \wedge s \geq v_1 \wedge \dots \wedge s \geq u_n \wedge s \geq v_n$):

superposition right:

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t \mid s|_p = l \wedge l > r \wedge l > \Gamma' \wedge s > t \wedge s > \Gamma}$$

superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma, s \simeq t \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta \mid s|_p = l \wedge l > r \wedge l > \Gamma' \wedge s > t \wedge s \geq \Gamma, \Delta}$$

equality resolution:

$$\frac{\Gamma, s \simeq t \rightarrow \Delta}{\Gamma \rightarrow \Delta \mid s = t \wedge s \geq \Gamma, \Delta}$$

where in both superposition rules $s|_p$ is required not to be a variable.

3.5. EXAMPLE. Consider the lexicographic path ordering generated by the precedence $h \succ_{\mathcal{F}} a \succ_{\mathcal{F}} f \succ_{\mathcal{F}} g \succ_{\mathcal{F}} b$. In the following inference

$$\frac{\begin{array}{l} g(x) \simeq x \rightarrow f(a, x) \simeq f(x, x) \\ g(x) \simeq x \rightarrow h(f(x, x)) \simeq h(y) \mid f(a, x) = f(a, g(y)) \wedge h(f(a, g(y))) > h(y) \wedge \\ f(a, x) > f(x, x) \wedge f(a, x) > g(x) \wedge f(a, x) > x \end{array}}{\rightarrow h(f(a, g(y))) \simeq h(y)}$$

the constraint of the conclusion is satisfiable: using the properties of the ordering and solving the unification problem, the constraint can be simplified into

$$x = g(y) \wedge a > x$$

which has, for instance, the solution $\{y \mapsto b, x \mapsto g(b)\}$.

On the other hand, the following inference is not needed

$$\frac{\begin{array}{l} \rightarrow f(x, x) \simeq f(a, x) \\ \rightarrow f(a, x) \simeq h(z) \mid f(x, x) = f(g(y), z) \wedge f(g(y), z) > h(z) \wedge \\ f(x, x) > f(a, x) \end{array}}{\rightarrow f(g(y), z) \simeq h(z)}$$

since the constraint of the conclusion has no solution; it can be simplified to

$$x = g(y) \wedge x = z \wedge y \geq h(z) \wedge x > a$$

which implies $y \geq h(g(y))$. Note that the equality constraint and the ordering constraint considered separately are both satisfiable but their conjunction is not. \square

Let us also remark that, at the non-ground level, several terms in a premise C may be involved in paramodulation inferences; for a term t it may be the case that for some ground instances $C\sigma$ the term $t\sigma$ is the maximal one, and for other instances it is not.

3.4. Completeness without constraint inheritance

There are several possible treatments for the constrained clauses generated by the inference system \mathcal{H} . The classical view is to deal only with unconstrained clauses. Conclusions of the form $C \mid s = t \wedge OC$, for some ordering constraint OC , are then immediately converted into $C\sigma$ where $\sigma = mgu(s, t)$. This strategy will be called here \mathcal{H} *without constraint inheritance*, in contrast with other possibilities which will be introduced later on.

Of course, the clause $C\sigma$ has to be generated only if the constraint $s = t \wedge OC$ is satisfiable in $\mathcal{T}(\mathcal{F})$, where $=$ is interpreted as the syntactic equality relation \equiv , and $>$ as the given reduction ordering \succ . If \succ is the lexicographic path ordering (LPO) the satisfiability of such constraints is decidable [Comon 1990, Nieuwenhuis 1993] (see Section 7 of this chapter). But traditionally in the literature weaker approximations by non-global tests are used; for example, inference systems are sometimes expressed with local conditions of the form $r \not\geq l$ when in our framework we have $l > r$ as a part of the global constraint OC . Note that such weaker approximations do not lead to unsoundness, but only to the generation of unnecessary (for completeness) clauses.

In the following, we call a set of (unconstrained) Horn clauses S *closed under \mathcal{H} without constraint inheritance* if $D\sigma \in S$ for all inferences by \mathcal{H} with premises in S and conclusion $D \mid s = t \wedge OC$ such that $s = t \wedge OC$ is satisfiable and $\sigma = mgu(s, t)$.

3.6. THEOREM. *The inference system \mathcal{H} is refutation complete without constraint inheritance for Horn clauses.*

PROOF. Let S be a set of Horn clauses closed under \mathcal{H} without constraint inheritance such that $\Box \notin S$. The proof is very similar to the one for \mathcal{G} ; we exhibit a model R^* for S . We proceed again by induction on \succ_c , but now the role of the ground clauses in the proof for \mathcal{G} is played by all *ground instances* of clauses in S , and the generation of rules in R from these ground instances is the same as for \mathcal{G} . Now we derive a contradiction from the existence of a minimal (w.r.t. \succ_c) *ground instance* $C\sigma$ of a clause C in S such that $R^* \not\models C\sigma$. The cases considered are the same ones as well, again depending on the occurrences in $C\sigma$ of its maximal term $s\sigma$.

The only difference lies in the *lifting* argument, which is the same in all cases and is hence analyzed here for only one of them: C is $\Gamma, s \simeq t \rightarrow \Delta$ and $s\sigma \succ t\sigma$. Since $R^* \not\models C\sigma$, we have $s\sigma \simeq t\sigma \in R^*$ and since R is convergent, $s\sigma$ must be reducible by some rule $l\sigma \Rightarrow r\sigma \in R$, generated by a clause C' of the form $\Gamma' \rightarrow l \simeq r$. (Note that, since we assume that there are no name clashes between the variables of C and C' , we can consider that the instances of C and of C' under consideration are both by the same ground σ .) Now we have $s\sigma|_p = l\sigma$, and there are two possibilities:

An inference. $s|_p$ is a non-variable position of s .

Then there exists an inference by superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma, s \simeq t \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta \quad | \quad s|_p = l \wedge l > r \wedge l > \Gamma' \wedge s > t \wedge s \geq \Gamma, \Delta}$$

whose conclusion $D \mid T$ has an instance $D\sigma$ (i.e., σ is a solution of T) such that $C\sigma \succ_c D\sigma$, where $R^* \not\models D\sigma$, contradicting the minimality of $C\sigma$.

Lifting. $s|_{p'}$ is a variable x for some prefix p' of p .

Then $p = p' \cdot p''$ for some p'' , and $x\sigma|_{p''}$ is $l\sigma$. Now let σ' be the ground substitution with the same domain as σ but where $x\sigma' = x\sigma[r\sigma]_{p''}$ and $y\sigma' = y\sigma$ for all other variables y . Then $R^* \not\models C\sigma'$ and $C\sigma \succ_c C\sigma'$, contradicting the minimality of $C\sigma$. \square

3.5. General clauses

In this section general clauses are considered, i.e., clauses that may have several equations in their succedents. For this purpose, the inference system \mathcal{H} is adapted. In order to restrict the amount of inferences to be performed, it is desirable to preserve the property of \mathcal{H} that for each ground clause (or instance) C , only one literal of C is involved in superposition inferences with C . Since now the maximal term of C may occur in more than one equation in the succedent, it is decided that among these equations the one whose other side is maximal will be used. This leads to the notion of maximal and strictly maximal equations in C . In order to express maximality and strict maximality of equations as constraints, we use the following notation. The constraint $gr(s \simeq t, \Delta)$ expresses that the equation $s \simeq t$, i.e., the multiset $\{s, t\}$, is strictly greater, w.r.t. the multiset extension of \succ , than all equations $u \simeq v$ in Δ . For each $u \simeq v$ this condition $s \simeq t \succ u \simeq v$ can be expressed for instance by the constraint:

$$s > u \wedge (s \geq v \vee t \geq v) \vee s > v \wedge (s \geq u \vee t \geq u) \vee \\ t > u \wedge (s \geq v \vee t \geq v) \vee t > v \wedge (s \geq u \vee t \geq u)$$

Similarly, the constraint $greq(s \simeq t, \Delta)$ expresses that $s \simeq t \succ u \simeq v$ for all $u \simeq v$ in Δ . The full inference system \mathcal{I} for general clauses is

superposition right:

$$\frac{\Gamma' \rightarrow l \simeq r, \Delta' \quad \Gamma \rightarrow s \simeq t, \Delta}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t, \Delta', \Delta \quad | \quad s|_p = l \wedge \\ l > r \wedge l > \Gamma' \wedge gr(l \simeq r, \Delta') \wedge \\ s > t \wedge s > \Gamma \wedge greq(s \simeq t, \Delta)}$$

superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r, \Delta' \quad \Gamma, s \simeq t \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta', \Delta \quad | \quad s|_p = l \wedge l > r \wedge l > \Gamma' \wedge gr(l \simeq r, \Delta') \wedge s > t \wedge greq(s \simeq t, \Gamma \cup \Delta)}$$

equality resolution:

$$\frac{\Gamma, s \simeq t \rightarrow \Delta}{\Gamma \rightarrow \Delta \quad | \quad s = t \wedge greq(s \simeq t, \Gamma \cup \Delta)}$$

equality factoring:

$$\frac{\Gamma \rightarrow s \simeq t, s' \simeq t', \Delta}{\Gamma, t \simeq t' \rightarrow s \simeq t', \Delta \quad | \quad s = s' \wedge s > t \wedge s > \Gamma \wedge greq(s \simeq t, \Delta \cup \{s' \simeq t'\})}$$

where as in the Horn case in both superposition rules $s|_p$ is not a variable.

Here the superposition rules and the equality resolution rule play the same role as their counterparts in the inference system \mathcal{H} . The equality factoring rule is new. Intuitively, it expresses that, if s and s' are syntactically equal, and t and t' are semantically equal, then the two equations in the succedent express the same information, and one of them can be omitted.

3.7. EXAMPLE. Consider the lexicographic path ordering generated by the precedence $f \succ_{\mathcal{F}} g \succ_{\mathcal{F}} h$ and the following inference by superposition right

$$\frac{\rightarrow g(z) \simeq h(z) \quad \rightarrow f(g(x), y) \simeq g(x), f(g(x), y) \simeq y}{\rightarrow f(h(z), y) \simeq g(x), f(g(x), y) \simeq y \quad | \quad g(x) = g(z) \wedge g(z) > h(z) \wedge f(g(x), y) > g(x) \wedge gr(f(g(x), y) \simeq g(x), \{f(g(x), y) \simeq y\})}$$

where $gr(f(g(x), y) \simeq g(x), \{f(g(x), y) \simeq y\})$ can be simplified into $g(x) > y$. Now, simplifying the rest of the constraint, the conclusion of the inference can be written as

$$\rightarrow f(h(z), y) \simeq g(x), f(g(x), y) \simeq y \quad | \quad x = z \wedge g(x) > y$$

Below an overview of the new aspects for the completeness proof of \mathcal{I} with respect to \mathcal{H} is given. For simplicity, only the ground case is considered; lifting to clauses with variables is analogous to what was done for \mathcal{H} . First, a new condition is added in the generation of the rewrite system R for a set of clauses S (see Section 3.1) and the second condition is adapted in order to select the strictly maximal positive equation that produces the rule:

3.8. DEFINITION. Let S be a set of ground clauses and let C be a clause in S . Then $\text{Gen}(C) = \{l \Rightarrow r\}$, and C is said to *generate* the rule $l \Rightarrow r$, if, and only if, C is of the form $\Gamma \rightarrow l \simeq r, \Delta$ and

1. $R_C^* \not\models C$
2. $l \succ r$, $l \succ \Gamma$, and $l \simeq r \gg u \simeq v$ for all $u \simeq v$ in Δ
3. l is irreducible by R_C
4. $R_C^* \not\models r \simeq t'$ for every $l \simeq t' \in \Delta$

where $R_C = \bigcup_{C \succ_c D} \text{Gen}(D)$. In all other cases $\text{Gen}(C) = \emptyset$. Finally, R denotes the set of all rules generated by clauses of S , that is, $R = \bigcup_{D \in S} \text{Gen}(D)$.

The proof of Lemma 3.2 can be easily adapted to show that here again R is convergent and that if $R_C^* \models C$ then $R^* \models C$. In a very similar way, it can be shown that the new conditions force clauses generating rules to have only one positive literal satisfied by the interpretation:

3.9. LEMMA. *If a clause C of the form $\Gamma \rightarrow l \simeq r, \Delta$ generates the rule $l \Rightarrow r$ then $R^* \models \Gamma$ and $R^* \not\models \Delta$.*

3.10. THEOREM. *The inference system \mathcal{I} is refutation complete for general clauses.*

PROOF. Since lifting is done as for \mathcal{H} , here we only extend the proof for the ground case \mathcal{G} . There is one additional case due to the new conditions for generating rules in R . The other cases of the proof for \mathcal{G} are straightforwardly adapted by using lemma 3.9 to show that the conclusion of the required inference is not satisfied by the model.

The new case is: C is of the form $\Gamma \rightarrow s \simeq t, \Delta$, with $s \succ t, \Gamma$ and $s \simeq t$ is maximal in Δ , and it has not generated a rule because there is an equation $s \simeq t'$ in Δ such that $R_C^* \models t \simeq t'$ (note that this case includes also the case in which $s \simeq t$ is maximal in Δ , but not strictly maximal).

Then, with $\Delta = s \simeq t', \Delta'$, there exists an inference by equality factoring

$$\frac{\Gamma \rightarrow s \simeq t, s \simeq t', \Delta}{\Gamma, t \simeq t' \rightarrow s \simeq t', \Delta}$$

whose conclusion D is such that $C \succ_c D$ and $R^* \not\models D$, contradicting the minimality of C . \square

3.6. Selection of negative equations

The inference system \mathcal{I} includes strong ordering restrictions: roughly, a superposition inference is needed only if the terms involved are maximal sides of maximal equations in their respective premises, and even strictly maximal in case they occur in positive equations. But more constraints can be imposed. If a clause C has a non-empty antecedent, it is possible to arbitrarily *select* exactly one of its negative

equations. Then completeness is preserved even if C is not used as left premise of any superposition inference and the only inferences involving C are equality resolution or superposition left on its selected equation.

The inference system S (for selection) for general clauses is defined to consist of the four rules of inference system I where for all premises of the inference rules no negative equation has been selected, plus the following two additional rules, where the selected equations have been underlined:

superposition left on a selected equation:

$$\frac{\Gamma' \rightarrow l \simeq r, \Delta' \quad \Gamma, \underline{s \simeq t} \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta', \Delta \quad | \quad s|_p = l \wedge l > r \wedge l > \Gamma' \wedge gr(l \simeq r, \Delta') \wedge s > t}$$

equality resolution on a selected equation:

$$\frac{\Gamma, \underline{s \simeq t} \rightarrow \Delta}{\Gamma \rightarrow \Delta \quad | \quad s = t}$$

where, as usual in superposition rules, $s|_p$ is not a variable.

Note that an adequate selection strategy gives us a strictly more restrictive inference system: among the set of maximal negative equations, just select one of them, and select no equation if this set is empty. It is clear that in the inference system \mathcal{I} all maximal equations of the antecedent are eligible for superposition left or equality resolution, whereas in S only the selected one is eligible.

The intuition behind selection is, roughly, that a clause with negative equations does not need to contribute to the deduction process until its whole antecedent has been proved from other clauses, and in particular one can require the selected equation to be proved first.

In practice one can select for example always a maximal equation (under some arbitrary ordering) of the antecedent. Selecting always a negative equation, whenever there is one, leads in the Horn case to the so-called *positive unit literal strategies*, that is, the left premise of superposition inferences is always a positive unit clause [Dershowitz 1991, Nieuwenhuis and Nivela 1991]. For general clauses eager selection leads to *positive strategies*, where the left premise is always a positive clause, i.e., it has only positive literals. Adapting the proof of completeness of Theorem 3.10 to this framework with selection is an easy exercise: it suffices to consider that clauses with selected equations generate no rules.

3.7. Merging paramodulation and perfect models

There is an alternative to the equality factoring inference rule, which is *merging paramodulation* rule plus *ordered factoring* [Bachmair and Ganzinger 1994b]. The inference system \mathcal{M} consists of the paramodulation rules and the equality resolution rule of \mathcal{I} , plus the following two rules:

merging paramodulation:

$$\frac{\Gamma' \rightarrow l \simeq r, \Delta' \quad \Gamma \rightarrow s \simeq t, s' \simeq t', \Delta}{\Gamma', \Gamma \rightarrow s \simeq t[r]_p, s' \simeq t', \Delta', \Delta \mid t|_p = l \wedge s = s' \wedge l > r \wedge l > \Gamma' \wedge gr(l \simeq r, \Delta') \wedge s > t \wedge s > \Gamma \wedge greg(s \simeq t, \Delta \cup \{s' \simeq t'\})}$$

ordered factoring:

$$\frac{\Gamma \rightarrow s \simeq t, s' \simeq t', \Delta}{\Gamma \rightarrow s \simeq t, \Delta \mid s = s' \wedge t = t' \wedge s > t \wedge s > \Gamma \wedge greg(s \simeq t, \Delta)}$$

where in the merging paramodulation rule $t|_p$ is not a variable.

The completeness proof for the resulting inference system \mathcal{M} can be obtained by exactly the same construction for the rewrite system R and the same cases as for \mathcal{I} , except that where before equality factoring was needed, now either merging paramodulation or ordered factoring apply.

An important property of the inference system \mathcal{M} is related to the following. For \mathcal{G} and \mathcal{H} , it is not difficult to see that the model R^* constructed from S is (isomorphic to) the unique minimal Herbrand model of S : it is a Herbrand model, as we have shown, and it is minimal, since all rules of R are logical consequences of S . This turns out to be very useful in applications to inductive theorem proving, see [Comon 2001] (Chapter 14 of this Handbook).

It is well-known that if S contains some non-Horn axiom, then in general a unique minimal Herbrand model of S no longer exists. For example, if $S \equiv \{p \vee q\}$ then both the models $\{p\}$ and $\{q\}$ are minimal. The total reduction ordering \succ on ground literals provides a way to single out one of the minimal models, the so-called *perfect model* (of S and \succ). The perfect model is the minimal one with respect to the (multi)set extension \succ_{mul}^{-1} of \succ^{-1} . If $S \equiv \{p \vee q\}$ where $p \succ q$ then $\{q\} \succ_{mul}^{-1} \{p\}$ and hence $\{p\}$ is the perfect model (see [Bachmair and Ganzinger 1991] for details).

Now it turns out that the model R^* obtained for sets of clauses closed under \mathcal{M} is indeed the perfect one, which is not the case for the inference system \mathcal{I} as shown in the following example⁷:

3.11. EXAMPLE. Assume we have $a \succ b \succ c \succ d$ and the following two clauses

$$\begin{aligned} &\rightarrow b \simeq d \\ &\rightarrow a \simeq b, a \simeq c \end{aligned}$$

Then the closure w.r.t. \mathcal{I} only introduces the new clause

$$b \simeq c \rightarrow a \simeq c$$

⁷By Leo Bachmair, private communication.

and a number of tautologies (that are not involved in the model construction). Therefore $R = \{b \Rightarrow d, a \Rightarrow b\}$, and the model $R^* = I$ is $\{b \simeq d, a \simeq b, a \simeq d\}$.

On the other hand, the closure by \mathcal{M} produces the clause

$$\rightarrow a \simeq d, a \simeq c$$

apart from other clauses that are not relevant for the generation of R . In this case $R = \{b \Rightarrow d, a \Rightarrow c\}$, and the model $R^* = M$ is $\{b \simeq d, a \simeq c\}$.

Now we have that $I \succ_{mul}^{-1} M$, since after removing $b \simeq d$ in both sets $a \simeq d \succ^{-1} a \simeq c$, i.e., I is not minimal. \square

In logic programming, perfect models give semantics for programs with negation (as failure), and the ordering \succ is usually induced from the way non-Horn clauses are written: one positive atom is written in the head of the clause, and the other ones are written negatively in the tail. For instance, $p \vee q$ can be written $p : - \neg q$ or $q : - \neg p$. Heads are made big in the ordering. If the resulting ordering is well-founded then the program has a perfect model. Roughly, a logic program with negation is called (locally) *stratified* if there is some well-founded ordering on ground atoms such that for all ground instances of clauses the head is bigger than every negative atom in the tail, and bigger than or equal to every positive atom in the tail [Przymusiński 1988]. Local stratification is too strong a condition for the existence of a perfect model, and it has been relaxed into *weak stratification*, where only ground instances contributing to the model need to fulfill the requirements [Przymusińska and Przymusiński 1990]. These ideas are generalized and extended to arbitrary programs with equality in [Bachmair and Ganzinger 1991].

4. Saturation procedures

The completeness results presented until now only apply to closure procedures, that is, deduction procedures which compute the closure of an initial set of clauses under a given inference system, without considering simplification or deletion techniques. However, such techniques are well-known to be crucial for efficiency in paramodulation-based theorem proving. In this section we study their compatibility with refutation completeness.

4.1. Redundancy in practice

Let us first give some examples of practical simplification and deletion methods. Most provers apply these methods in two possible contexts. The first one, usually called *forward* redundancy elimination, is applied to new clauses immediately after they are obtained by an inference. For example, the conclusion of an inference can be simplified by rewriting it using other clauses before storing it. On the other hand, *backward* techniques are the ones applied to existing clauses, using newer ones that have been generated later on.

4.1. EXAMPLE. Consider the lexicographic path ordering generated by the precedence $f \succ_{\mathcal{F}} a \succ_{\mathcal{F}} b$ and the following two equations whose maximal side is written underlined:

1. $\underline{f(a, x)} \simeq x$
2. $\underline{f(x, a)} \simeq f(x, b)$

There is a superposition inference with conclusion

$$f(a, b) \simeq a$$

to which forward simplification can be applied by rewriting it with equation 1 into

$$b \simeq \underline{a}$$

Adding the result to the set, we obtain:

1. $\underline{f(a, x)} \simeq x$
2. $\underline{f(x, a)} \simeq f(x, b)$
3. $\underline{a} \simeq b$

Now, by backward simplification using the new equation 3, equation 2 can be simplified into the tautology $f(x, b) \simeq f(x, b)$. The elimination of this tautology is another backward redundancy step. Furthermore, equation 1 can be simplified using 3 into

$$\underline{f(b, x)} \simeq x$$

Hence the final set will only contain the equations

3. $\underline{a} \simeq b$
4. $\underline{f(b, x)} \simeq x$

□

In this section it is explained how redundancy elimination methods like the ones used in this example can be treated uniformly in the context of *saturation* procedures. Let us first give some informal intuition. The notion of saturation w.r.t. a given inference system \mathcal{I} generalises the one of closure w.r.t. \mathcal{I} : roughly, a set of clauses S is *saturated* if S is closed under \mathcal{I} *up to redundant inferences*. Refutation completeness then means that the empty clause \square is in S for every unsatisfiable saturated set of clauses S .

A procedure like the one of the previous example can be seen as a procedure computing a saturated set. Such a *saturation* procedure will be modelled by a *derivation*, a possibly infinite sequence of sets of clauses where each set can be obtained from the previous one in two possible ways: either by adding a clause or by removing a clause.

Two abstract notions of redundancy will play an essential role in saturation: one for clauses and one for inferences. Here we first explain them informally.

Roughly, a clause C is redundant w.r.t. a set S if C follows from clauses in S that are smaller than C . Redundant clauses correspond to the ones that are removed in a derivation. This abstract notion provides a useful means for proving the completeness of the inference system in combination with concrete practical (e.g., backward) redundancy elimination techniques.

Similarly, an inference will be redundant if its conclusion follows from clauses that are smaller than its maximal premise. In a derivation, conclusions of redundant inferences need not to be added. This abstract notion of redundant inference covers several powerful concrete forward redundancy techniques.

In the remainder of this section these ideas are formally developed and explained.

4.2. Redundancy and saturation in the ground case

In this section, as a simple example, saturation is described in detail for the case of ground clauses. Hence in this section all clauses (denoted by C, D, \dots) and sets of clauses (denoted by S) are assumed to be ground. A two-stage approach is followed. First a “static” point of view is considered: it is proved that unsatisfiable saturated sets contain the empty clause. This involves the notion of redundant inference. After this, the “dynamic” problem of how to compute such saturated sets is considered, which is where the concepts of derivation and of redundant clauses are needed.

4.2.1. the static view

In the previous section we built a model R^* for any set S closed under \mathcal{I} and such that $\square \notin S$. Now our aim is to weaken the closedness requirement as much as possible into some notion of saturatedness. This could of course be done by defining a set S to be saturated whenever $R^* \models S$, but this would not be very useful in practice, since this (global) semantic property can almost never be checked. In order to find a useful practical notion of saturatedness, one can analyse the kind of inferences that are really needed in the proof showing that $R^* \models S$ for closed sets: the ones in which the rightmost premise is the minimal clause that is false in R^* . This leads us to the following.

We denote by $S^{<C}$ the set of all D in S such that $C \succ_c D$. An inference with maximal premise C and conclusion D is *redundant with respect to a set S* if $S^{<C} \models D$. A set of clauses S is *saturated* with respect to an inference system Inf if every inference of Inf with premises in S is redundant with respect to S .

4.2. THEOREM. *Let S be a set of ground clauses that is saturated with respect to \mathcal{I} . Then $R^* \models S$ if, and only if, $\square \notin S$, and hence S is unsatisfiable if, and only if, $\square \in S$.*

PROOF. The only difference with Theorem 3.10 is that now a contradiction has to be obtained from the fact that the inferences between clauses in S are redundant instead of considering that the conclusion is in S . Let us show it for an inference by

superposition right. In this case the minimal counterexample C has not generated a rule because its maximal term is reducible by a rule generated by a clause C' . Then the following inference by superposition right is considered where C' is the left premise, C is the right one and D is the conclusion.

$$\frac{\Gamma' \rightarrow l \simeq r, \Delta' \quad \Gamma \rightarrow s \simeq t, \Delta}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t, \Delta', \Delta}$$

As in Theorem 3.10, by using Lemma 3.9 from $R^* \not\models C$ we can infer that $R^* \not\models D$. But on the other hand, since the inference is redundant we have $S^{\prec c} \models D$, and by minimality of C we have $R^* \models D$ which is a contradiction. \square

4.2.2. the dynamic view: computing saturated sets

The previous theorem states that, instead of computing sets closed under the inference system, it suffices to saturate them. Therefore, now practical methods for computing saturated sets are defined. These methods are formalized by the notion of *derivation*, a sequence of sets of clauses where each time the next set is obtained by either adding some logical consequence or removing some *redundant* clause.

A ground clause C is *redundant* with respect to a set of ground clauses S if $S^{\prec c} \models C$. A *derivation* is a (possibly infinite) sequence of sets of clauses S_0, S_1, \dots where each S_{i+1} is either $S_i \cup \{C\}$, for some C such that $S_i \models C$, or $S_i \setminus \{C\}$, for some C that is redundant with respect to S_i . Clauses belonging, from some i on, to all S_k with $k > i$, are called *persistent*. The set S_∞ is the set of persistent clauses, defined $S_\infty = \cup_i \cap_{k>i} S_k$.

A nice property of the general notion of redundancy presented above is given by the following lemma. It states that all non-persistent clauses occurring in the derivation are redundant w.r.t. the set of persistent ones.

4.3. LEMMA. *Let S_0, S_1, \dots be a derivation and let C be a clause in $(\cup_i S_i) \setminus S_\infty$. Then C is redundant w.r.t. S_∞ .*

PROOF. We proceed by induction on C w.r.t. \succ_c . Since $C \notin S_\infty$ there is some S_j , s.t. C is redundant w.r.t. S_j , which implies that $S_j^{\prec c} \models C$, and hence by induction hypothesis $S_\infty^{\prec c} \models C$. \square

It is easy to see that simplification by demodulation fits into the notion of derivation. For example, simplifying $P(f(a))$ into $P(a)$ with the equation $f(a) \simeq a$ is modeled by first adding $P(a)$, and then removing $P(f(a))$ which has become redundant in the presence of $P(a)$ and $f(a) \simeq a$.

4.4. EXAMPLE. Consider the lexicographic path ordering generated by the precedence $P \succ_{\mathcal{F}} Q \succ_{\mathcal{F}} f \succ_{\mathcal{F}} a$. The table:

Refutation S	Comments	Derivation S_0, S_1, \dots
1. $\rightarrow \underline{Q(a)}$ 2. $\underline{Q(a)} \rightarrow f(a) \simeq a$ 3. $\underline{P(a)} \rightarrow$ 4. $\rightarrow P(f(a))$	Initial set of clauses	S_0
1. $\rightarrow Q(a)$ 2. $Q(a) \rightarrow f(a) \simeq a$ 3. $P(a) \rightarrow$ 4. $\rightarrow \underline{P(f(a))}$ 5. $\rightarrow \underline{f(a) \simeq a}$	Inf. 5 from 1 and 2	$S_0 \models (\rightarrow f(a) \simeq a)$ $S_1 = S_0 \cup \{ \rightarrow f(a) \simeq a \}$
1. $\rightarrow Q(a)$ 2. $Q(a) \rightarrow f(a) \simeq a$ 3. $\underline{P(a)} \rightarrow$ 5. $\rightarrow f(a) \simeq a$ 6. $\rightarrow \underline{P(a)}$	Simplif. 4 to 6 with 5	$S_1 \models (\rightarrow P(a))$ $S_2 = S_1 \cup \{ \rightarrow P(a) \}$ <hr/> $\rightarrow P(f(a))$ is redundant w.r.t. S_2 $S_3 = S_2 \setminus \{ \rightarrow P(f(a)) \}$
1. $\rightarrow Q(a)$ 2. $Q(a) \rightarrow f(a) \simeq a$ 3. $P(a) \rightarrow$ 5. $\rightarrow f(a) \simeq a$ 6. $\rightarrow P(a)$ 7. \square	Inf. 7 from 6 and 3	$S_3 \models \square$ $S_4 = S_3 \cup \{ \square \}$

represents a derivation performed by a theorem prover that is based on the strict superposition calculus and applies simplification by demodulation. The first column contains the set of ground clauses at every step of the refutation and the second one contains some explanations about the current step. In the third column the sets of the derivation are given and the changes justified. In the first column, the underlined subterms are the ones involved in the next inference. \square

If our aim is to obtain refutation complete theorem proving procedures by com-

puting derivations, and in the limit, to obtain a saturated set, then a notion of *fairness* is required. It roughly says that no inference π should be postponed forever: either some premise of π disappears at some point of the derivation, or else π has to become redundant at some point. One way of forcing π to become redundant is by adding its conclusion: for every ground inference π by our inference systems, always its conclusion is smaller than its maximal premise⁸ and hence π is trivially redundant w.r.t. a set S containing its conclusion D (since D follows from a clause of S that is smaller than the maximal premise, namely D itself).

4.5. DEFINITION. A derivation S_0, S_1, \dots is *fair* with respect to an inference system Inf if for every inference π of Inf with premises in S_∞ there is some $j \geq 0$ s.t. π is redundant with respect to S_j .

Now we can prove that fair derivations compute (in the limit) saturated sets and generate the empty clause if and only if the initial set is unsatisfiable.

4.6. THEOREM. If S_0, S_1, \dots is a fair derivation with respect to Inf , then S_∞ is saturated with respect to Inf , and hence if Inf is \mathcal{I} , then S_0 is unsatisfiable if, and only if, $\square \in S_j$ for some j . Furthermore, if S_0, S_1, \dots, S_n is a fair derivation then S_n is saturated and logically equivalent to S_0 .

PROOF. First we prove that S_∞ is saturated with respect to Inf . By fairness, all inferences with premises in S_∞ are redundant in some S_j and hence, by lemma 4.3, redundant in S_∞ , which implies that S_∞ is saturated.

Second, if Inf is \mathcal{I} , since S_∞ is saturated with respect to \mathcal{I} , by theorem 4.2 S_∞ is unsatisfiable if, and only if, $\square \in S_\infty$. Since, definition of derivation, S_0 is logically equivalent to all S_i with $i \geq 0$ and, by lemma 4.3, to S_∞ as well, S_0 is unsatisfiable if, and only if, $\square \in S_\infty$ and hence $\square \in S_j$ for some j .

Finally if S_0, S_1, \dots, S_n then $S_\infty = S_n$ and hence S_n is saturated. \square

Now, what can be done in practice to ensure fairness? On the one hand, it is needed that after adding the conclusion of an inference, the inference becomes redundant. As said, for the inference systems presented in this chapter, this is indeed the case. In order to capture forward simplification we also want the inference to become redundant if we add the simplification of the conclusion. Indeed, with this notion of redundancy of inferences any clause smaller than the maximal premise can be used to simplify the conclusion into a smaller clause.

On the other hand, this has to be done for all inferences with persistent premises. But since it is not possible to know, at a certain point S_i , whether a given premise is going to be persistent or not, some means should be provided ensuring that no inference with persistent premises is postponed infinitely many times. In most

⁸For inference systems not satisfying this property, an inference should be redundant as well w.r.t. a set S if its conclusion is in S (and not only if its conclusion follows from clauses in S smaller than its maximal premise). Another possibility is to relax the notion of *fairness* (that will be introduced in a moment), requiring that either the conclusion of π belongs to some S_j or else π is redundant in S_j .

implementations this is achieved by periodically considering all inferences with the clause whose *size* (in number of symbols) is smallest. If a certain clause persists then it will eventually be considered, since there are only finitely many clauses with smaller size.

4.3. Non-ground saturation procedures

For the non-ground case, the definitions for redundancy of inferences and clauses of the previous section are straightforwardly extended (roughly, C or π is redundant if all its ground instances are) and the notions of derivation and saturatedness do not change. Here we consider saturation without constraint inheritance, that is, the S_i occurring in derivations are sets of clauses without constraints, and if fairness requires a non-ground inference π with conclusion $D \mid s = t \wedge OC$ to become redundant in some S_j , then this is done by adding $D\sigma$ to S_j , where $\sigma = mgu(s, t)$.

In the following, $gnd(C)$ denotes the set of all ground instances of a clause C , and if S is a set of clauses then $gnd(S)$ denotes $\bigcup_{C \in S} gnd(C)$.

Let π be an inference with premises C_1, \dots, C_n and conclusion $D \mid T$. Then a *ground instance* $\pi\sigma$ of the inference π is an inference with premises $C_1\sigma, \dots, C_n\sigma$ and conclusion $D\sigma$ for some ground σ such that $\sigma \models T$. An inference π is *redundant* with respect to a set S if all its ground instances are redundant with respect to $gnd(S)$. Note that an inference whose conclusion has an unsatisfiable constraint is redundant since it has no ground instances.

4.7. EXAMPLE. Consider the lexicographic path ordering generated by the precedence $P \succ_{\mathcal{F}} f \succ_{\mathcal{F}} h \succ_{\mathcal{F}} g \succ_{\mathcal{F}} a$ and the following set S of equations whose maximal sides are written underlined:

1. $\frac{g(x)}{\quad} \simeq x$
2. $\frac{h(a, z)}{\quad} \simeq z$
3. $\frac{f(x, \underline{h(x, y)})}{\quad} \simeq g(y)$
4. $\frac{\underline{f(a, z)}}{\quad} \simeq z$

The inference between 2 and 3 can be shown redundant w.r.t. S using rewriting as follows. It has the conclusion

$$f(x, z) \simeq g(y) \mid h(a, z) = h(x, y) \wedge h(a, z) > z \wedge f(x, h(x, y)) > g(y)$$

Once it is checked that the constraint is satisfiable, the most general unifier $\{x \mapsto a, z \mapsto y\}$ of the unification problem in the constraint is applied to the conclusion, obtaining:

$$f(a, y) \simeq g(y)$$

It has to be shown that all its ground instances, which are of the form $f(a, t) \simeq g(t)$, follow from instances of S smaller than the corresponding instance of the

maximal premise, which is $f(a, h(a, t)) \simeq g(t)$. This can be done by rewriting: both sides of $f(a, y) \simeq g(y)$ rewrite into y using equations 4 and 1.

As another example of forward simplification, assume the set consists only of equations 1, 2 and 3. The inference between 2 and 3 we have seen generates $f(a, y) \simeq g(y)$, which by forward simplification with 1 produces equation 4. \square

It is easy to show, by a similar lifting argument as the one used in Theorem 3.6, that the non-ground version of Theorem 4.2 holds.

4.8. THEOREM. *Let S be a set of clauses that is saturated with respect to \mathcal{I} . Then, S is unsatisfiable if, and only if, $\square \in S$.*

Now we can again focus on the problem of how to compute (non-ground, this time) saturated sets. For this purpose, in this context a clause C is *redundant* with respect to a set S if all its ground instances are redundant with respect to $\text{gnd}(S)$.

The notions of non-ground derivation, persistence and fairness are defined exactly as in the ground case. The non-ground versions of Lemma 4.3 and Theorem 4.6 can be proved in a similar way.

4.9. THEOREM. *If S_0, S_1, \dots is a fair derivation with respect to Inf , then S_∞ is saturated with respect to Inf , and hence, if Inf is \mathcal{I} , then S_0 is unsatisfiable if, and only if, $\square \in S_j$ for some j . Furthermore, if S_0, S_1, \dots, S_n is a fair derivation then S_n is saturated and logically equivalent to S_0 .*

4.10. EXAMPLE. Let us now consider a more complicated example showing the power of the notion of redundancy for inferences, where moreover the generated ordering constraints are not ignored like in Example 4.7, but play a crucial role.

Consider the transitivity axiom for a predicate p :

$$p(x, y) \wedge p(y, z) \rightarrow p(x, z)$$

Consequences by superposition left (or resolution) of this clause are:

$$p(x, y) \wedge p(y, z) \wedge p(z, u) \rightarrow p(x, u)$$

$$p(x, y) \wedge p(y, z) \wedge p(z, u) \wedge p(u, w) \rightarrow p(x, w)$$

...

We first show that these consequences are not redundant clauses in the presence of the transitivity axiom. An instance of $p(x, y) \wedge p(y, z) \wedge p(z, u) \rightarrow p(x, u)$ of the form

$$p(a, b) \wedge p(b, c) \wedge p(c, d) \rightarrow p(a, d)$$

only follows from instances of the transitivity axiom

$$p(b, c) \wedge p(c, d) \rightarrow p(b, d) \tag{4.1}$$

$$p(a, b) \wedge p(b, d) \rightarrow p(a, d) \tag{4.2}$$

$$p(a, b) \wedge p(b, c) \rightarrow p(a, c) \tag{4.3}$$

$$p(a, c) \wedge p(c, d) \rightarrow p(a, d) \tag{4.4}$$

in two possible ways: from (4.1,4.2) or from (4.3,4.4). However, if $b \succ_{\mathcal{F}} a \succ_{\mathcal{F}} c \succ_{\mathcal{F}} d$ then in both cases the instances used are not smaller than the instance that has to be proved redundant. Therefore, the clause $p(x, y) \wedge p(y, z) \wedge p(z, u) \rightarrow p(x, u)$ is not redundant. But we can prove that the two possible resolution inferences producing it from the transitivity axiom are indeed redundant.

One of the two inferences is

$$\frac{p(x, y) \wedge p(y, u) \rightarrow p(x, u) \quad p(y, z) \wedge p(z, u) \rightarrow p(y, u)}{p(x, y) \wedge p(y, z) \wedge p(z, u) \rightarrow p(x, u) \mid p(y, u) > p(x, u) \wedge p(y, u) > p(x, y) \wedge p(y, u) > p(y, z) \wedge p(y, u) > p(z, u)}$$

in which the unifier has already been applied. The constraint of the conclusion can be simplified into $y > x \wedge u > z \wedge y > z$. Now the ground instances of the conclusion that indeed satisfy this constraint follow from smaller instances of the set 4.1–4.4, i.e., the inference is redundant.

Another difficult question is: how to find in practice, and automatically, the appropriate clauses and their instances that allow us to prove such redundancies? In the Saturate system [Nivela and Nieuwenhuis 1993, Ganzinger, Nieuwenhuis and Nivela 1999], several such concrete techniques are implemented. For this concrete example, Saturate proves the redundancies automatically by *clausal rewriting* combined with LPO constraint solving. \square

4.4. More general notions of redundancy for clauses

As said, the notion of redundancy of clauses given in the previous section together with the notion of derivation can capture simplification methods like demodulation by rewriting. However, it cannot capture useful methods like *subsumption* in its full generality. For instance, a clause $P(a)$ (for some constant a) cannot be proved redundant w.r.t. a set containing $P(x)$, since only strictly smaller instances can be used in the redundancy proof.

To overcome this problem, the notion of redundancy of clauses can be made more powerful by applying not only smaller instances but also equal instances in the redundancy proof.

We denote by $S^{\leq c}$ the set of all D in S such that $C \succeq_c D$. Then a clause C is *non-strictly redundant* w.r.t. a set of clauses S if $\text{gnd}(S)^{\leq d} \models C$ for all D in $\text{gnd}(C)$. Note that it is equivalent to the requirement that for all D in $\text{gnd}(C)$ either $\text{gnd}(S)^{< d} \models C$ or $D \in \text{gnd}(S)$. This means that one only needs to care about all those ground instances of C that do not belong to S . It is easy to see that this notion of redundancy covers subsumption.

4.11. EXAMPLE. The clause $P(a)$ is redundant w.r.t. $\{P(x)\}$, since $P(a) \in \text{gnd}(P(x))$. But if the signature under consideration is fixed, then it is possible to go beyond.

Assume \mathcal{F} is $\{a, f, Q, P\}$, let C be the clause $Q(x) \vee P(x)$, and let S be the set of clauses $\{P(f(y)), Q(a) \vee P(a)\}$.

Then C can be proved non-strictly redundant w.r.t. the set of clauses S as follows. The ground instance $Q(a) \vee P(a)$ of C is in S and is hence redundant w.r.t. S . The remaining ground instances of C are of the form $Q(f(t)) \vee P(f(t))$ for some ground term t , which are redundant since $P(f(t)) \models Q(f(t)) \vee P(f(t))$ and $Q(f(t)) \vee P(f(t)) \succ_c P(f(t))$.

Note that some instances of C have been proved strictly redundant, i.e., using smaller instances w.r.t. \succ_c , and others have been proved non-strictly redundant, that is using equal instances in $\text{gnd}(S)$. \square

In this new setting with non-strict redundancy, the notion of derivation has to be slightly modified. If S_{i+1} is $S_i \setminus \{C\}$, now we require C to be (non-strictly) redundant w.r.t. $S_i \setminus \{C\}$, instead of w.r.t. S_i as before (otherwise all clauses C in S_i could be removed!).

Unfortunately, this stronger notion of redundancy for clauses has some side effects on the notion of fairness, mainly because there might be persistent ground instances that do not correspond to any persistent clause.

4.12. EXAMPLE. Assume $\mathcal{F} = \{a, P\}$ and the following derivation:

$$\begin{aligned} S_0 &= \{\neg P(x), P(x)\}, \\ S_1 &= \{\neg P(x), P(x), P(a)\}, \\ S_2 &= \{\neg P(x), P(a)\}, \\ S_3 &= \{\neg P(x), P(x), P(a)\}, \\ S_4 &= \{\neg P(x), P(x)\}, \\ &\dots \end{aligned}$$

The only instance of $P(x)$ is $P(a)$. Therefore $P(x)$ is redundant in the presence of $P(a)$, and vice versa, and hence the sequence S_0, S_1, \dots is indeed a derivation. The only persistent clause is $\neg P(x)$, and no inference between persistent clauses exists. Hence the empty clause will not be generated in this derivation. This problem is clearly due to the fact that fairness, as it was stated in the previous subsection for the weaker notion of redundancy of clauses, only requires inferences between persistent clauses to be considered. \square

In the previous example there is a clause $P(a)$, which is a *persistent ground instance*, i.e. a ground clause C such that from some k on, C belongs to all $\text{gnd}(S_i)$ with $i > k$, which is not an instance of any persistent clause. Therefore a simple way to overcome this problem is to modify the notion of fairness by requiring in addition that, roughly, the set of persistent ground instances is covered by the set of persistent clauses. This idea is formalised as follows.

4.13. DEFINITION. The set $G_\infty = \bigcup_i \bigcap_{k>i} \text{gnd}(S_i)$ is the set of *persistent ground instances*. A derivation S_0, S_1, \dots is *ground fair* with respect to an inference system Inf if $\text{gnd}(S_\infty) \supseteq G_\infty$ and all inferences of Inf with premises in S_∞ are redundant w.r.t. S_j for some j .

Ground fairness can be achieved in practical theorem provers by associating to each clause a counter indicating its “level” of non-strict redundancy steps, and

forbidding such non-strict redundancy steps beyond a certain level. In most implementations the problem of the previous example is avoided automatically, since, as said, fairness is achieved by periodically considering all inferences with the smallest clause with respect to size. If a certain ground instance persists, then at any point it is an instance of some clause with smaller or equal size, and hence it will eventually be considered, because there are only finitely many such clauses with smaller size.

The non-ground version of Lemma 4.3 can be proved for non-strict redundancy in the same way as before, using the fact that $gnd(S_\infty) \supseteq G_\infty$. Theorem 4.9 holds as well.

4.5. Computing with saturated sets

In practice, it is sometimes possible to obtain a finite saturated set S_n (not containing the empty clause) for a given input S_0 . In this case its satisfiability has been proved. Let us give an example.

4.14. EXAMPLE. Consider the lexicographic path ordering generated by the precedence $P \succ_{\mathcal{F}} Q \succ_{\mathcal{F}} f \succ_{\mathcal{F}} g \succ_{\mathcal{F}} a$. Table 1 represents a finite derivation that terminates with a saturated set. The concrete redundancy method applied is simplification by demodulation. The first column contains the set of clauses at each step of the derivation. In the second column the sets of the derivation are given and the changes justified. In the first column, the underlined terms are the ones involved in the next inference.

The final set S_4 is saturated since all inferences with premises in S_4 are redundant:

1. the inferences between 3 and 4 and between 2 and 6 are redundant since their conclusions are in S_4 (and hence they follow from clauses smaller than the maximal premise).
2. the inference between 5 and 6 is also redundant, although its conclusion is not in S_4 . Let us show it. The inference has premises $g(g(y)) \simeq g(y)$ (note that we have renamed the variables of 5 to avoid name clashes) and $Q(g(x)) \rightarrow P(g(x))$, with the unifier $\sigma = \{x \mapsto g(y)\}$. The conclusion is $Q(g(g(y))) \rightarrow \overline{P(g(y))}$, which can be rewritten by 5 into $Q(g(y)) \rightarrow P(g(y))$, which is smaller than 6σ and belongs (up to renaming of variables) to S_4 , and hence the inference is redundant. \square

The remainder of this section is on the applications of such finite saturation derivations. On the one hand, the existence of a finite saturated set S not containing the empty clause implies that its consistency has been proved. Consistency proving has many applications and is also closely related to inductive theorem proving, as shown in [Comon 2001] (Chapter 14 of this Handbook).

But on the other hand, theorem proving in theories expressed by saturated sets S of axioms is also interesting because more efficient proof strategies become (refutationally) complete. For example, it is clear from the previous section that when saturating $S \cup S'$, for some S' , inferences all whose premises belong to S are redundant in $S \cup S'$, for the following reason. Since S is saturated, these inferences are

Derivation S_0, S_1, \dots	Comments
1. $Q(g(g(x))) \rightarrow P(g(x))$ 2. $P(g(a)) \rightarrow$ 3. $\rightarrow \underline{f(x, y)} \simeq g(x)$ 4. $\rightarrow \underline{f(x, a)} \simeq g(g(x))$	Initial set of clauses S_0
1. $\underline{Q(g(g(x)))} \rightarrow P(g(x))$ 2. $P(g(a)) \rightarrow$ 3. $\rightarrow f(x, y) \simeq g(x)$ 4. $\rightarrow f(x, a) \simeq g(g(x))$ 5. $\rightarrow \underline{g(g(x))} \simeq g(x)$	Infer 5 from 3 and 4 $S_1 = S_0 \cup \{5\}$
2. $\underline{P(g(a))} \rightarrow$ 3. $\rightarrow f(x, y) \simeq g(x)$ 4. $\rightarrow f(x, a) \simeq g(g(x))$ 5. $\rightarrow g(g(x)) \simeq g(x)$ 6. $Q(g(x)) \rightarrow \underline{P(g(x))}$	Simplify 1 into 6 with 5 $S_1 \models 6$ $S_2 = S_1 \cup \{6\}$ 1 is redundant w.r.t. $S_2 \setminus \{1\}$ $S_3 = S_2 \setminus \{1\}$
2. $\underline{P(g(a))} \rightarrow$ 3. $\rightarrow f(x, y) \simeq g(x)$ 4. $\rightarrow f(x, a) \simeq g(g(x))$ 5. $\rightarrow g(g(x)) \simeq g(x)$ 6. $Q(g(x)) \rightarrow \underline{P(g(x))}$ 7. $Q(g(a)) \rightarrow$	Infer 7 from 2 and 6 $S_3 \models 7$ $S_4 = S_3 \cup \{7\}$

Table 1: A finite derivation terminating with a saturated set

redundant in S , and hence as well in any set containing S , because the redundancy notions are easily shown to be monotonic in this sense.

This leads to the completeness of the set-of-support strategy, where S' is the set of support. This strategy is complete for standard binary resolution, but is incomplete in general for ordered inference systems and also for paramodulation ([Snyder and Lynch 1991] describe a lazy paramodulation calculus that is complete with set of support).

4.6. Completion as an instance of saturation

In some cases, depending on the syntactic properties of the given finite saturated set S , decision procedures for the given theory are obtained. This is the case for instance for saturated sets of unit equations E , where the saturation process behaves like unfailing Knuth-Bendix completion. Clearly, simplification by rewriting and removing tautologies $s \simeq s$ (and other more refined techniques) fit into the redundancy notions. Furthermore, indeed rewriting with the final saturated set provides a decision procedure for the word problem.

Let \succ be a total reduction ordering, and let E be a set of unit equations. Furthermore, let \rightarrow_E be the ordered rewrite relation (remember that \rightarrow_E is the smallest monotonic relation on terms such that $s\sigma \rightarrow_E t\sigma$ whenever $s \simeq t$ is in E and $s\sigma \succ t\sigma$).

4.15. THEOREM. *If E is a set of unit equations that is saturated w.r.t. \mathcal{H} and \succ , then every ground term s has a unique normal form $nf(s)$ w.r.t. \rightarrow_E . Furthermore, for every pair of ground terms s and t , $E \models s \simeq t$ if, and only if, $nf(s) \equiv nf(t)$.*

PROOF. It is shown that every ground term s (possibly with new Skolem constants for its variables) can be rewritten into the unique minimal (w.r.t. \succ) representative of its E -congruence class. By induction w.r.t. \succ , it suffices to prove the reducibility w.r.t. \rightarrow_E of non-minimal s . Let u be this minimal representative of the congruence class of s . Since $s \succ u$, the only inference rule that can be applied in a refutation of $s \simeq u \rightarrow$ are strict superposition left steps on s with some positive equation $l \simeq r$ of E . But then s is reducible by rewriting with $l \simeq r$, since $s|_p = l\sigma$ for some p . \square

In fact, similar results apply as well to the other forms of saturatedness that will be introduced later on in this chapter (modulo equational theories, with constraint inheritance).

Decision procedures are also obtained for the ground case. For the ground Horn inference system \mathcal{G} applied with eager selection, clearly each inference of $l \simeq r$ on a clause D produces a smaller clause D' . Furthermore, D is a logical consequence of the smaller clauses $l \simeq r$ and D' , i.e., D has become redundant and can be removed. Hence after each inference, the clause set decreases w.r.t. the well-founded multiset extension of the clause ordering and hence the process terminates, thus deciding satisfiability.

4.16. THEOREM. *Superposition with selection decides the satisfiability of sets of ground Horn clauses.*

Furthermore, a decision procedure for the satisfiability of sets of arbitrary ground clauses is obtained by first transforming into Horn clauses (where $SUC \vee A_1 \vee \dots \vee A_n$ is split into the disjunction of sets S_i of the form $S \cup C \vee A_i$; then S is satisfiable if some of the S_i is).

4.17. EXAMPLE. Note, however, that ground saturation procedures without redundancy do not always terminate, in spite of the fact that only smaller ground clauses are created in a well-founded ordering. Consider an LPO with $a \succ_{\mathcal{F}} f \succ_{\mathcal{F}} b$ and the initial two ground equations

$$\begin{array}{ll} 1. & f(\underline{a}) \simeq a \\ 2. & f(b) \simeq \underline{a} \end{array}$$

Then infinitely many equations i , for $i > 2$ are generated by superposition at the underlined subterm between equation 1 and equation $i - 1$

$$\begin{array}{ll} 3. & f(f(b)) \simeq \underline{a} \\ 4. & f(f(f(b))) \simeq \underline{a} \\ 5. & f(f(f(f(b)))) \simeq \underline{a} \\ & \vdots \end{array}$$

□

Other syntactic restrictions on non-equational saturated sets S that are quite easily shown to produce decision procedures include *reductive* Horn clauses (also called conditional equations) or *universally reductive* general clauses [Bachmair and Ganzinger 1994b]. In such cases, the non-redundant inferences in a refutation of $S \cup G$ for certain classes of ground clauses G only produce new smaller ground clauses and saturation terminates by a similar argument as in the ground case. This kind of ideas provide several directions in which the previous two theorems can be generalized (see also Section 8.2 of this chapter).

4.7. Extended signatures

When applying the results we have seen so far for computing with sets $S \cup S'$ where S is saturated, one aspect has to be considered carefully: what happens if new (e.g. Skolem) symbols appear in S' ?

4.18. EXAMPLE. Suppose S is the following set of unit equations:

$$\{\rightarrow f(x) \simeq a, \quad \rightarrow g(a) \simeq a\}$$

under a lexicographic path ordering with the precedence $g \succ_{\mathcal{F}} a \succ_{\mathcal{F}} f$.

This set is saturated with respect to *the given signature*: the only possible inference with the two equations of S not needed, since its conclusion, $g(f(x)) \simeq a \mid a > f(x)$, has an unsatisfiable constraint $a > f(x)$ because a is the smallest constant symbol. From the rewriting and Knuth-Bendix completion point of view, S being saturated with respect to a given signature means that \rightarrow_S is confluent for rewriting terms built over this signature, i.e., it is *ground confluent*, which is a weaker property than general confluence.

Now let us try to prove, for instance, that $S \models \forall y \ g(f(y)) \simeq a$. After negating and Skolemizing the goal $G = g(f(b)) \simeq a \rightarrow$ is obtained, which has to be refuted, where b is a new Skolem constant. Then, under the new extension of the precedence $g \succ_{\mathcal{F}} a \succ_{\mathcal{F}} f \succ_{\mathcal{F}} b$, the set-of-support strategy fails: no inferences can be computed between equations in S and G , but $S \cup \{G\}$ is inconsistent. Equivalently, from the rewriting point of view, $S \models G$ but G is in normal form with respect to \rightarrow_S .

This incompleteness is due to the fact that S is not saturated with respect to the new signature (note that a is no longer the smallest constant symbol). If S is instead saturated with respect to *extended* signatures then the inference with conclusion $g(f(x)) \simeq a \mid a > f(x)$ should be performed, since the constraint $a > f(x)$ is satisfiable in some extension of the signature. Then this incompleteness problem does not appear. \square

From the previous example we learn that for some applications it is necessary to solve the ordering constraints under extended signatures (see Section 7 for details on ordering constraint solving). Similar incompleteness problems appear if we apply redundancy methods that rely on the given signature like, for instance, the one used in example 4.11.2.

Let us now consider the combination of two finite sets of clauses S_1 and S_2 (built over \mathcal{F}_1 and \mathcal{F}_2 resp.) that are saturated with respect to \succ_1 and \succ_2 respectively. Then an extension of the set-of-support-strategy applies: no inferences have to be considered in which all premises are in S_1 or all premises in S_2 . Therefore, if $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$ then $S_1 \cup S_2$ is saturated.

Again here it is needed that S_1 and S_2 are saturated under the semantics in which the satisfiability of the constraints has been considered with respect to extended signatures, or at least with respect to a signature containing $\mathcal{F}_1 \cup \mathcal{F}_2$. Otherwise, situations similar to the example above can again appear.

Furthermore, it is necessary to find an ordering \succ with all the required properties containing both \succ_1 and \succ_2 . If \succ_1 and \succ_2 are two orderings of the same family of path orderings and this family is stable under extensions of the precedence (e.g. RPO is such a family) then one can define a precedence $\succ_{\mathcal{F}_1 \cup \mathcal{F}_2}$ extending $\succ_{\mathcal{F}_1}$ and $\succ_{\mathcal{F}_2}$ (whenever $\succ_{\mathcal{F}_1}$ and $\succ_{\mathcal{F}_2}$ are not contradictory, that is, $f, g \in \mathcal{F}_1 \cap \mathcal{F}_2$ and $f \succ_{\mathcal{F}_1} g$ implies $f \succ_{\mathcal{F}_2} g$). This produces a total extension. See [Rubio 1995] for related results on combining arbitrary orderings.

5. Paramodulation with constrained clauses

In this section we develop strategies where the ordering and/or equality restrictions of the inferences are kept in constraints and inherited between clauses. As explained in Section 1, this produces a further pruning of the search space. For simplicity reasons, first only Horn clauses are considered and at the end of the section the extension to general constrained clauses is outlined.

5.1. Equality constraint inheritance: basic strategies

We now analyse the first constraint-based restriction of the search space: the so-called *basicness restriction*, where superposition inferences on subterms created by unifiers on ancestor clauses are not performed. This restriction is conveniently expressed by inheriting the equality constraints without applying (or even computing) any unifiers. Hence from now on we consider sets of *constrained* clauses, rather than unconstrained ones, as in the previous sections.

5.1. DEFINITION. In the following, we call a set of constrained Horn clauses S *closed under \mathcal{H} with equality constraint inheritance* if $D \mid T_1 \wedge \dots \wedge T_n \wedge s = t$ is in S whenever $C_1 \mid T_1, \dots, C_n \mid T_n$ are clauses in S and there is an inference by \mathcal{H} with premises C_1, \dots, C_n and conclusion $D \mid s = t \wedge OC$ such that the constraint $T_1 \wedge \dots \wedge T_n \wedge s = t \wedge OC$ is satisfiable.

This strategy is incomplete in general: the closure under \mathcal{H} with equality constraint inheritance of an unsatisfiable set of constrained Horn clauses needs not contain the empty clause.

5.2. EXAMPLE. Let \succ be the lexicographic path ordering where $a \succ_{\mathcal{F}} b$. Consider the following unsatisfiable clause set S :

1. $\rightarrow a \simeq b$
2. $\rightarrow P(x) \mid x = a$
3. $P(b) \rightarrow$

S is clearly closed under \mathcal{H} with equality constraint inheritance, since no inferences by \mathcal{H} that are compatible with the constraint of the second clause can be made. We have $a \succ_{\mathcal{F}} b$ and hence the first clause could only be used by superposing a on some non-variable subterm, while superposition left (i.e., resolution) between 2 and 3 leads to a clause with an unsatisfiable constraint $x = a \wedge b = x$. However, S does not contain the empty clause. This incompleteness is due to the fact that the usual lifting arguments, like the ones in Theorem 3.6, do not work here, since they are based on the existence of *all* ground instances of the clauses. Note that this is not the case here: the only instance of the second clause is $P(a)$, whereas the lifting argument in Theorem 3.6 requires the existence of the instance $P(b)$. \square

Fortunately, the strategy is complete for what we will call *well-constrained* sets of clauses, which turn out to be adequate for many practical situations. A key idea in this context is the following (quite intuitive) notion of *irreducible ground substitution*. Let R be a ground rewrite system contained in the given ordering \succ (that is, $l \succ r$ for all rules $l \Rightarrow r$ of R). A ground substitution σ is *reducible* by R if $x\sigma$ is reducible by R for some $x \in \text{Dom}(\sigma)$; if there is no such x then σ is *irreducible*. Furthermore, if S is a set of constrained clauses, then $\text{irred}_R(S)$ is its set of *irreducible instances*, that is, the set of ground instances $C\sigma$ of clauses $C \mid T$ in S such that σ is a solution of T and $x\sigma$ is irreducible by R for all $x \in \text{vars}(C)$.

5.3. DEFINITION. A set of constrained clauses S is *well-constrained* if either there are no clauses with equality predicates in S or else for all R contained in \succ we have $\text{irred}_R(S) \cup R \models S$.

5.4. EXAMPLE. (Example 5.2 continued) The clause set S of the previous example is not well-constrained: if R is $\{a \Rightarrow b\}$ then the instance $P(a)$ of the second clause is not a logical consequence of $\text{irred}_R(S) \cup R$ (in fact, the second clause has no irreducible instances for this R). \square

Let us give some more intuition behind the definition of well-constrained sets. For clauses without equality predicates, the situation is clear: all such sets are well-constrained (this is why logic programming without equality is compatible with arbitrary constraint systems).

Now let us consider clause sets S including equality predicates. First, note that if S is a well-constrained set, so is its closure w.r.t. any sound inference system, since the property of well-constrainedness is preserved under the addition of logical consequences. Second, it is not difficult to see that if all clauses in S have only tautologic constraints then S is well-constrained: every instance $C\sigma$ is either in $\text{irred}_R(S)$, or else σ is reducible by R . Then σ can be reduced into a “normal form” σ' , where $C\sigma'$ is in $\text{irred}_R(S)$, and we have $\text{irred}_R(S) \cup R \models C\sigma$.

But there are other non-trivial examples of well-constrained sets.

5.5. EXAMPLE. Let \succ be the lexicographic path ordering where $g \succ_{\mathcal{F}} a \succ_{\mathcal{F}} f \succ_{\mathcal{F}} b$. Then, constrained clauses like $g(x, x) \simeq b \mid a > x$ may appear in well-constrained sets, since the variable x is not “lower bounded”: as for unconstrained clauses, for all σ the term $x\sigma$ can be reduced into a “normal form” $x\sigma'$, where $g(x\sigma', x\sigma') \simeq b$ is in $\text{irred}_R(S)$, and hence we have $\text{irred}_R(S) \cup R \models g(x\sigma, x\sigma)$. Here $g(x, x) \simeq b \mid a > x$ denotes the infinite set of clauses of the form $g(f^n(b), f^n(b)) \simeq b$ for $n \geq 0$, that is, $g(b, b) \simeq b$, $g(f(b), f(b)) \simeq b$, $g(f(f(b)), f(f(b))) \simeq b \dots$ Note that such (in this case even non-regular) tree languages cannot be captured by standard first-order clauses. \square

Furthermore, it will become clear from the completeness proof below that the notion of well-constrained clause could be modified in order to capture more cases by not considering *all* R contained in \succ , but only those R whose rules could be generated in the model generation technique applied to the given clause set. Then,

one can know in advance that certain (e.g., constructor) terms will be irreducible w.r.t. such R . Here we have not done this in order to keep the definition of well-constrainedness simple.

The refutation completeness of \mathcal{H} for well-constrained clause sets S can now be established by applying a simple variant of the model generation technique. Before we give the formal proof, let us explain the main ideas. Let S be a set of well-constrained clauses that is closed under \mathcal{H} with equality constraint inheritance, and assume $\square \notin S$. As always, we show that then S is satisfiable by generating a rewrite system R for S (in a similar way as before) and then proving that $R^* \models S$.

For this purpose, we first show that $R^* \models \text{irred}_R(S)$ like in Theorem 3.6, but where the lifting case never needs to be applied (since we only consider the set of irreducible instances of S). Once we have $R^* \models \text{irred}_R(S)$, then also $R^* \models S$, since of course $R^* \models R$ and by well-constrainedness of S (where well-constrainedness is required only with respect to the particular R that has been generated) we have $\text{irred}_R(S) \cup R \models S$ (note that if there are no equality literals in S then $\text{irred}_R(S)$ coincides with S).

5.6. THEOREM. *The inference system \mathcal{H} is refutation complete with equality constraint inheritance for well-constrained sets S of Horn clauses.*

PROOF. Let S be closed under \mathcal{H} with equality constraint inheritance. Again we build a model R^* for S whenever $\square \notin S$. As said, we prove that $R^* \models \text{irred}_R(S)$, which implies $R^* \models S$ by well-constrainedness.

We build R as for Theorem 3.6, but now only the irreducible (w.r.t. R_C) instances of S contribute to its construction: a ground instance C of the form $\Gamma \rightarrow l \Rightarrow r$ in $\text{irred}_{R_C}(S)$ generates the rule $l \Rightarrow r$ of R if the usual conditions (i), (ii) and (iii) apply.

Now again we derive a contradiction from the existence of a minimal (w.r.t. \succ_c) ground instance $C\sigma \in \text{irred}_R(S)$ for some $C \mid T \in S$, where σ is a solution of T , such that $R^* \not\models C\sigma$. Again we consider several cases, depending on the occurrences in $C\sigma$ of its maximal term $s\sigma$. Let us analyse only the case where C is $\Gamma, s \simeq t \rightarrow \Delta$ and $s\sigma \succ t\sigma$. Since $R^* \not\models C\sigma$, we have $R^* \models s\sigma \simeq t\sigma$, and hence the term $s\sigma$ is reducible by some rule $l\sigma \Rightarrow r\sigma \in R$, generated by an instance $C'\sigma$ of some $C' \mid T'$, where C' is of the form $\Gamma' \rightarrow l \simeq r$.

Now we have $s\sigma|_p = l\sigma$, and, since σ is irreducible by R , the only possibility is now that $s|_p$ is a non-variable position of s . Then there exists an inference by superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma, s \simeq t \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta \quad |s|_p = l \wedge l > r \wedge l > \Gamma' \wedge s > t \wedge s \geq \Gamma, \Delta}$$

whose conclusion has an instance $D\sigma$ where σ is a solution of $T \wedge T' \wedge s|_p = l \wedge l > r \wedge l > \Gamma' \wedge s > t \wedge s \geq \Gamma, \Delta$ such that $C\sigma \succ_c D\sigma$ and where $R^* \not\models D\sigma$. Furthermore, $D\sigma \in \text{irred}_R(S)$: indeed $x\sigma$ is irreducible by R for all variables $x \in \text{vars}(D)$. This is clearly the case if $x \in \text{vars}(C)$. For $x \in C'$, there

are two cases: if $x \equiv l$ then $x \notin \text{vars}(D)$ since $l\sigma \succ r\sigma, \Gamma'\sigma$; if $x \neq l$ then $x\sigma$ is irreducible w.r.t. $R_{C'}$ by construction of R , and hence also w.r.t. R , since for all rules $l' \Rightarrow r' \in R \setminus R_{C'}$ we have $l' \succeq l\sigma \succ x\sigma$ and hence such rules cannot reduce $x\sigma$. Altogether, this contradicts the minimality of $C\sigma$. \square

5.2. Ordering constraint inheritance

The proof ideas used for equality constraint inheritance apply as well to ordering constraint inheritance or to a combination of both.

A set of constrained Horn clauses S is *closed under \mathcal{H} with ordering constraint inheritance* if $(D \mid T_1 \wedge \dots \wedge T_n \wedge OC) \sigma$ is in S whenever $C_1 \mid T_1, \dots, C_n \mid T_n$ are clauses in S and there is an inference by \mathcal{H} with premises C_1, \dots, C_n and conclusion $D \mid s = t \wedge OC$ such that $\sigma = \text{mgu}(s, t)$ and the constraint $T_1 \wedge \dots \wedge T_n \wedge s = t \wedge OC$ is satisfiable.

5.7. THEOREM. *The inference system \mathcal{H} is refutation complete with ordering constraint inheritance for well-constrained sets S of Horn clauses.*

A set of constrained Horn clauses S is *closed under \mathcal{H} with equality and ordering constraint inheritance* if $D \mid T_1 \wedge \dots \wedge T_n \wedge s = t \wedge OC$ is in S whenever $C_1 \mid T_1, \dots, C_n \mid T_n$ are clauses in S and there is an inference by \mathcal{H} with premises C_1, \dots, C_n and conclusion $D \mid s = t \wedge OC$ such that the constraint $T_1 \wedge \dots \wedge T_n \wedge s = t \wedge OC$ is satisfiable.

5.8. THEOREM. *The inference system \mathcal{H} is refutation complete with equality and ordering constraint inheritance for well-constrained sets S of Horn clauses.*

5.3. Basic paramodulation

It is possible to further restrict the inference system \mathcal{H} with constraint inheritance, at the expense of weakening the ordering restrictions. Roughly, the improvement comes from the possibility of moving the inserted right hand side (denoted by r in our superposition rules) in conclusions to the constraint part, thus *blocking* this term for further inferences. On the other hand, paramodulations take place only *with* the maximal term, like in superposition, but *on* both sides of the equation containing the maximal term. More precisely, the inference rule of (ordered, basic) paramodulation right then becomes:

ordered paramodulation right:

$$\frac{\Gamma' \rightarrow l \simeq r \quad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[x]_p \simeq t \mid x=r \wedge s|_p=l \wedge l>r \wedge l>\Gamma' \wedge (s>\Gamma \vee t>\Gamma)}$$

where $s|_p$ is not a variable, and x is a new variable. The inference rule for paramodulation left is defined analogously. It is clear that these inference rules are an improvement upon superposition only when applied (at least) with inheritance of the

part $x = r$ of the equality constraint, since otherwise the advantage of blocking r is lost.

The completeness proof is an easy extension of the previous results by the model generation method. It suffices to modify the rule generation by requiring, when a rule $l \Rightarrow r$ is generated, that both l and r are irreducible by R_C , instead of only l as before, and to adapt the proof of Theorem 5.6 accordingly, which is straightforward. We refer to [Bachmair, Ganzinger, Lynch and Snyder 1995] for a deeper discussion of this form of basic paramodulation.

5.4. Saturation for constrained clauses

In this section the redundancy notions for constrained clauses and inferences are defined. The idea is similar to how it was done for unconstrained clauses with variables, except that here, as in the proofs of refutation completeness of \mathcal{H} with constraint inheritance, the ground instances are replaced by the set of irreducible (w.r.t. some R) ground instances. These definitions are of a rather theoretical nature. Practical sufficient conditions for them are given in [Nieuwenhuis and Rubio 1995].

In the following, $C \mid T$ and $D \mid T$ (or sometimes simply C, D) denote constrained clauses, S denotes a set of constrained clauses, and R denotes sets of ground rewrite rules included in \succ .

First, to get some intuition, let us look at an example showing that the usual simplification techniques are not compatible with constraint inheritance, even if the initial set has no constraints at all. For simplicity, we consider here only equality constraint inheritance, and a simplification step in which $f(g(a))$ is simplified into $f(b)$ by demodulation with the instance $g(a) \simeq b$ of $g(x) \simeq b \mid x = a$. Note that this is the natural extension of the standard method of simplification by rewriting with unconstrained equations, which, as we have seen, does not lead to incompleteness when no constraints are inherited.

5.9. EXAMPLE. Consider an LPO with $f \succ_{\mathcal{F}} g \succ_{\mathcal{F}} a \succ_{\mathcal{F}} b$ and the inconsistent set of four initial clauses:

1. $\rightarrow a \simeq b$
2. $\rightarrow f(g(x)) \simeq g(x)$
3. $\rightarrow f(g(a)) \simeq b$
4. $g(b) \simeq b \rightarrow$

Now we could generate a saturation process as follows:

5. $\rightarrow g(x) \simeq b \quad \mid x = a \quad (\text{by superposition of 3 on 2})$
6. $\rightarrow f(b) \simeq g(x) \quad \mid x = a \quad (\text{by superposition of 5 on 2})$
- 3'. $\rightarrow f(b) \simeq b \quad (\text{simplifying 3 by 5})$

Finally, the set $\{1, 2, 3', 4, 5, 6\}$ is closed under the inference rules, but the empty clause has not been generated. \square

Note that the problem is caused by the fact that, although the initial set is well-constrained, after applying the simplification step well-constrainedness is lost, and, as a consequence, refutation completeness. Therefore the redundancy methods should consider only irreducible instances, in order to be consistent with the techniques applied in the previous sections for constraint inheritance.

We denote by $\text{irred}_R(\pi)$ the set of ground instances $\pi\sigma$ of an inference π with constraint inheritance such that $C\sigma \in \text{irred}_R(C \mid T)$ for each $C \mid T$ that is premise or conclusion of π .

Then, an inference π is redundant in S if for every R compatible with \succ and for every $\pi\sigma \in \text{irred}_R(\pi)$ with premises C_1, \dots, C_n , maximal premise C and conclusion D , either $R \cup \text{irred}_R(S)^{\prec_{C_i}} \models C_i$ for some $i \in \{1 \dots n\}$ or $R \cup \text{irred}_R(S)^{\prec_C} \models D$.

Similarly, a constrained clause $C \mid T$ is redundant in S if, for every R compatible with \succ , $R \cup \text{irred}_R(S)^{\preceq_{C\sigma}} \models C\sigma$ for every $C\sigma \in \text{irred}_R(C \mid T)$. Note that non-strict redundancy of clauses (see Section 4.4 for details) is considered, which is crucial for showing that some powerful simplification methods based on constraints fit in this framework.

It is not difficult to see that in redundancy proofs one can use equations with tautologic constraints or constrained equations like $f(x) \simeq x \mid a > x$ for simplification by rewriting. But by means of constraints one can go beyond:

5.10. EXAMPLE. Let \succ be the lexicographic path ordering where $f \succ_{\mathcal{F}} a \succ_{\mathcal{F}} b$, and consider the set of equations

1. $f(x) \simeq a$
2. $f(b) \simeq b$

By analyzing its possible ground instances, equation 1 can be *split* into the one where x is b and the remaining instances. In the former case, 1 can be simplified with 2, and in the latter case the constraint $x \neq b$ can be added, obtaining, respectively, equations 3 and 4:

2. $f(b) \simeq b$
3. $b \simeq a$
4. $f(x) \simeq a \mid x \neq b$

By simplifying 4 with 3 we obtain

2. $f(b) \simeq b$
3. $b \simeq a$
5. $f(x) \simeq b \mid x \neq b$

Finally, 2 and 5 can be removed by adding 6

3. $b \simeq a$
6. $f(x) \simeq b$

\square

Related techniques are applied for paramodulation without any ordering restrictions (plus a certain kind of inferences inside constraints) in [Bourelly, Caferra and Peltier 1994].

We can now state refutation completeness, which is proved by combining the techniques of Theorems 5.6 and 4.8.

5.11. THEOREM. *Let S be a well-constrained set of clauses that is saturated w.r.t. \mathcal{I} with constraint inheritance. Then S is satisfiable if, and only if, $\square \notin S$.*

Instead of going into the details of derivations and fairness for constrained clauses, let us only remark here that the methodology explained for clauses without constraints in Section 4 produces results analogous to the ones of Theorem 4.9 for well-constrained sets of clauses.

5.5. General constrained clauses

When considering constraint inheritance for general clauses, the main proof method is the same as before. However, some additional details have to be handled, which make it altogether quite technical. For extending Theorem 5.6, the problems are caused by one case in the proof that has to be considered more carefully: the case where an instance $C\sigma$ of a constrained clause $C \mid T$ of the form $\Gamma \rightarrow x \simeq r, x \simeq r', \Delta \mid T$ generates a rule $x\sigma \Rightarrow r\sigma$ of R . Then, although $C\sigma$ is an instance with a substitution σ that is irreducible with respect to $R_{C\sigma}$, it is reducible with respect to R , since $x\sigma$ is reducible by the rule $x\sigma \Rightarrow r\sigma$ itself.

This situation has the following unpleasant consequences. If an inference with $C\sigma$ on another irreducible instance $C'\sigma$ is needed, it cannot be ensured any more that the corresponding instance $D\sigma$ of the conclusion $D \mid T''$ obtained from $C' \mid T'$ and $C \mid T$ is an irreducible instance, since D has $x \simeq r'$ in the succedent. Note that in the Horn case this cannot happen: if $x\sigma$ is the left hand side of the rule, then x cannot occur in the corresponding conclusion.

The problem is solved by refining the notion of irreducibility for these special variables occurring in an instance $C\sigma$. The problematic variables of $C\sigma$ are those variables that occur in the succedent and only in equations like $x \simeq r$ with $x\sigma \succ r\sigma$. For these variables $x\sigma$ is only required to be irreducible by rules smaller than the greatest $x\sigma \simeq r\sigma$ in $C\sigma$. With this notion of irreducibility, the proof of theorem 5.6 can be applied to general clauses, using the inference system \mathcal{I} and its corresponding rule generation as in Theorem 3.10. We refer to [Nieuwenhuis and Rubio 1995] for details.

5.12. THEOREM. *The inference system \mathcal{I} is refutation complete with equality and/or ordering constraint inheritance for well-constrained sets S of clauses.*

6. Paramodulation with built-in equational theories

In this section the paramodulation calculus is generalised to the case in which some of the initial axioms are considered as a built-in theory. In particular, the case in which the theory is expressed by a set E of equational axioms is considered.

There are different ways to extend paramodulation based inference systems for this purpose. The simplest one is by adding a new inference rule applying paramodulations *on* the equations of the theory (but not *with* them). An alternative to this inference rule is to associate to each clause the set of its *E-extended clauses* [Peterson and Stickel 1981], which are clauses obtained by adding to the maximal equation (if it is in the succedent) contexts coming from the equations in E . Then paramodulation is performed with the *E-extended clauses* as well. Due to the fact that many of these extended clauses can be shown to be redundant, this method seems to be less prolific than the first one.

In some interesting cases, like for abelian semigroups, that is, associative and commutative (AC) theories, the useful extended clauses can be easily characterized. Then it becomes possible as well to design specific inference rules instead of handling these extensions explicitly. This is the way most paramodulation calculi for the AC-case are expressed [Paul 1992, Rusinowitch and Vigneron 1995, Vigneron 1994, Nieuwenhuis and Rubio 1997] and in Section 6.2 (see also [Rubio 1996] for built-in semigroups, i.e. associative theories). This approach is considered as well for arbitrary *regular* theories in [Vigneron 1996].

Recent research concerns algebraic structures richer than abelian semigroups, like abelian groups [Stuber 1998a, Godoy and Nieuwenhuis 2000], cancellative abelian monoids [Ganzinger and Waldmann 1996], commutative rings [Stuber 1998b] or divisible torsion-free abelian groups [Waldmann 1998].

6.1. *E-compatible reduction orderings*

Many results in the literature on ordered paramodulation and superposition modulo E require (i) E -unification to be finitary, (ii) E -unifiability to be decidable, and (iii) the existence of an E -compatible total reduction ordering. In Section 6.3 we will describe a uniform framework in which the first two requirements turn out to be unnecessary by inheriting equality constraints. Only recently, in [Bofill, Godoy, Nieuwenhuis and Rubio 1999] it was proved that the third requirement can be dropped as well: E -compatible total reduction orderings, which were crucial in all previously existing completeness proofs completeness of ordered paramodulation calculi, are not needed for ordered paramodulation. The new results for paramodulation with non-monotonic orderings of [Bofill et al. 1999] may allow one to work with much simpler orderings. For example, in many cases one can normalize terms w.r.t. the theory E before comparing them by some total ordering on ground terms, thus obtaining a total, E -compatible, and well-founded ordering (that is not monotonic in general). However, the results for non-monotonic orderings have not been developed yet for working modulo equational theories, they are applicable only for

ordered paramodulation and not for superposition, and, moreover, they are not compatible with the usual redundancy elimination techniques. Hence it seems reasonable to expect that they will be used only in contexts where E -compatible total reduction orderings do not (or are not known to) exist.

Hence it is necessary to explore the possibilities of finding E -compatible reduction orderings for different theories E and to study in which cases these orderings can be E -total, i.e. total on the E -congruence classes. This is done in the remainder of this section. The following abbreviations will be used for equational axioms: C (commutativity), A (associativity), U (unit), I (idempotence) and D (distributivity).

First we will present some positive results for theories containing associativity and/or commutativity axioms. The easiest case is C, since RPO (see section 2.2) is a C-compatible reduction ordering if all commutative function symbols have a multiset status, and it is C-total under a total precedence if a lexicographic status is assigned to all other symbols. Similarly, in fact any *permutative* theory can be considered, that is, any theory E presented by axioms of the form $f(x_1, \dots, x_n) \simeq f(x_{\pi(1)}, \dots, x_{\pi(n)})$, where the x_1, \dots, x_n are distinct variables and π is some permutation of $1 \dots n$. If such f have multiset status, the ordering will be E -compatible. With a little more effort, it can be made total up to $=_E$ by a lexicographic combination with a second component⁹.

AC-axioms are present in many interesting theories, and hence AC-compatible orderings have received much more attention than any others. It turned out to be quite difficult to find AC-compatible reduction orderings, especially when AC-totality is also required. In fact, the first attempts were not total in general (see e.g. [Bachmair and Plaisted 1985, Ben-Cherifa and Lescanne 1987, Kapur, Sivakumar and Zhang 1990]). The first AC-compatible AC-total reduction ordering was exhibited in [Narendran and Rusinowitch 1991], while the first such ordering based on RPO appeared in [Rubio and Nieuwenhuis 1995] and further improvements on AC-orderings with RPO-scheme are developed in [Kapur and Sivakumar 1997, Rubio 1999]. For the A (associativity only) case, not many results have been developed. Of course, if A-totality is not required, any of the AC-orderings can be used. Otherwise, in [Rubio 1996], a way to obtain A-compatible A-total reduction orderings from AC-orderings is described. However, apparently some of the known RPO-like AC-total orderings could also be adapted to the A case directly. Finally, joining all the results one can obtain E -compatible E -total reduction orderings for theories E containing A-, C-, AC- and free symbols [Rubio 1994].

When considering other theories, fewer positive results can be obtained. U-compatible orderings cannot fulfill the subterm property, since if $+$ is a function symbol with unit 0 then $x + 0 =_U x$ and hence, by U-compatibility, $x + 0$ cannot

⁹In this second component, roughly, the multisets formed by the equivalence classes of permuting arguments are compared lexicographically. For example, if $a \succ_{\mathcal{F}} b$ and E consists of $f(x_1, x_2, x_3, x_4, x_5) \simeq f(x_1, x_3, x_2, x_4, x_5)$ and $f(x_1, x_2, x_3, x_4, x_5) \simeq f(x_1, x_2, x_3, x_5, x_4)$, then we can compare lexicographically sequences of multisets of arguments $\langle \{1st\}, \{2nd, 3rd\}, \{4th, 5th\} \rangle$, and $f(a, a, a, b, a) \succ f(a, a, b, a, a)$, since the multiset $\{a, a\}$ of the second and third argument of the first term is larger than the one of the second term, which is $\{b, a\}$.

be greater than x . This means that E -compatible *simplification* orderings do not exist if there are any such unit axioms in E . However, it is possible, as described in [Jouannaud and Marché 1992] and [Wertz 1992], to obtain an ACU-compatible *reduction* ordering from an AC-compatible reduction ordering, provided that all units are minimal in the given AC-ordering. But such a restriction has to be required by any E -compatible ordering such that E contains any unit axioms, i.e. $U \subseteq E$:

6.1. EXAMPLE. Let $+$ and $*$ be U-function symbols with units 0 and 1 respectively. Then if $0 \succ_E 1$, by monotonicity, $x + 0 \succ_E x + 1$, which implies, by E -compatibility (since $x + 0 =_U x$), $x \succ_E x + 1$, contradicting, by monotonicity, the well-foundedness of \succ_E . The symmetric case $1 \succ_E 0$ leads to the same contradiction. \square

This example shows us that only one unit is allowed if we are interested in E -totality. There may exist U-compatible reduction orderings that are U-total but which are not simplification orderings.

The case in which E contains some idempotence axiom is even worse, since then no E -compatible well-founded ordering \succ_E can be monotonic:

6.2. EXAMPLE. Let $*$ be an I-function symbol and let s and t be terms with $s \succ_E t$. Then if \succ_E is monotonic we have $s * s \succ_E t * s$ and hence, by E -compatibility (since $s * s =_I s$), $s \succ_E t * s$, which together with monotonicity contradicts well-foundedness. \square

Finally another interesting example is the presence of distributivity axioms. It is unknown whether there are, in general, D-compatible reduction orderings (and also E -compatible for a set E containing distributivity axioms). However, a well-known ACD-compatible ordering is the APO [Bachmair and Plaisted 1985], when there are no distribution *chains*.

6.2. Paramodulation modulo associativity and commutativity

Let us now consider the case of superposition with built-in associativity and commutativity for some function symbols.

In this section constraints are interpreted as follows: the ordering \succ interpreting $>$ is now an AC-compatible AC-total reduction ordering, while $=$ is interpreted as $=_{AC}$ (the AC-equality congruence).

The full inference system for general clauses modulo AC, called \mathcal{I}_{AC} includes the rules of \mathcal{I} (under the new semantics for $>$ and $=$) plus the following three specific rules:

AC-superposition right:

$$\frac{\Gamma' \rightarrow l \simeq r, \Delta' \quad \Gamma \rightarrow s \simeq t, \Delta}{\Gamma', \Gamma \rightarrow s[f(r, x)]_p \simeq t, \Delta', \Delta \quad \mid \quad \begin{array}{l} s|_p = f(l, x) \wedge \\ l > r \wedge l > \Gamma' \wedge gr(l \simeq r, \Delta') \wedge \\ s > t \wedge s > \Gamma \wedge gr(s \simeq t, \Delta) \end{array}}$$

AC-superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r, \Delta'}{\Gamma', \Gamma, s[f(r, x)]_p \simeq t \rightarrow \Delta', \Delta} \quad | \quad \begin{array}{l} \Gamma, s \simeq t \rightarrow \Delta \\ s|_p = f(l, x) \wedge \\ l > r \wedge l > \Gamma' \wedge gr(l \simeq r, \Delta') \wedge \\ s > t \wedge greq(s \simeq t, \Gamma \cup \Delta) \end{array}$$

AC-top-superposition:

$$\frac{\Gamma' \rightarrow l \simeq r, \Delta'}{\Gamma', \Gamma \rightarrow f(r', x') \simeq f(r, x), \Delta', \Delta} \quad | \quad \begin{array}{l} \Gamma \rightarrow s \simeq t, \Delta \\ f(l', x') = f(l, x) \wedge \\ l > r \wedge l > \Gamma' \wedge gr(l \simeq r, \Delta') \wedge \\ s > t \wedge s > \Gamma \wedge gr(s \simeq t, \Delta) \end{array}$$

In these rules, where x and x' are new variables, the term l can be restricted to be headed by the AC-symbol f . This can be expressed in the constraint language and added to the constraint. Let us define $u|_q$ to be a *maximal non- f subterm* of u if q is a position such that $top(u|_q) \neq f$ and $top(u|_{q'}) = f$ for all proper prefixes q' of q . Then, AC-top-superposition is only needed if l and l' are headed by f and share some maximal non- f subterm s but x and x' do not (some restrictions implied by this condition can be formulated in the constraint language, and hence some cases of failure of this condition can be detected as unsatisfiable constraints). Finally, the superposition inferences are needed only if $l|_p$ is non-variable (as usual), and AC-superposition is needed only if moreover $l|_p$ (which has an f as top symbol) is not immediately below another f . Some examples are given below.

The refutation completeness of \mathcal{I}_{AC} can be proved by introducing a notion of *extended instance* of a clause and then adapting the construction of the rewrite system R to work modulo AC and considering these extended instances for the generation of rules. We refer to [Nieuwenhuis and Rubio 1997] for the details.

6.3. THEOREM. *The inference system \mathcal{I}_{AC} is refutation complete without constraint inheritance with built-in AC-theories.*

6.3. Constraint inheritance and built-in theories

By inheriting equality constraints one can avoid one of the main drawbacks of working with built-in theories, namely the large cardinality of the set of unifiers for certain unification problems. For instance, there may be doubly exponentially many AC-unifiers for two terms [Domenjoud 1992] (in a sense, this is also an upper bound [Kapur and Narendran 1992]), and therefore as many conclusions in an inference; e.g., a minimal complete set for $x + x + x$ and $y_1 + y_2 + y_3 + y_4$ contains more than a million unifiers.

For proving refutation completeness with constraint inheritance it becomes necessary, as in the free case (see section 5.1), to consider irreducible instances. In this case the irreducibility notion for the fresh variables introduced by the three specific AC-inference rules needs to be refined. Again we refer to [Nieuwenhuis and Rubio 1997] for the details.

6.4. THEOREM. *The inference system \mathcal{I}_{AC} is refutation complete with constraint inheritance for well-constrained sets S of clauses with built-in AC-theories.*

As said, by inheriting equality constraints, the computation of unifiers and the generation of many conclusions in every inference becomes unnecessary. But it is possible to go beyond. One can deal, in an effective way, with theories E with an *infinitary* E -unification problem, i.e., theories where for some unification problems any complete set of unifiers is infinite. This is the case for theories containing only associativity axioms for some function symbols, which is developed in [Rubio 1996].

Finally, one can consider any built-in theory E , even when the E -unification problem is undecidable, if an adequate inference system and ordering are found (although these ideas require a further development for concrete E). Incomplete methods can then be used for detecting cases of unsatisfiable constraints, and only when a constrained empty clause $\square \mid T$ is found, one has to start (in parallel) a semidecision procedure proving the satisfiability of T . But note that for soundness only the equality constraint part of T needs to be proved satisfiable, since the inference rules are sound as well without ordering restrictions. This method is not only interesting if no decision procedure for the E -unification problem is available: incomplete methods can be more efficient and hence more effective in practice than complete ones.

7. Symbolic constraint solving

Equality constraints are also known as *unification problems*, since they generalize the notion of unification, which usually consists in solving one single equation. Due to the large amount of applications of unification in automated deduction, logic programming and, in general, in symbolic computation, equational constraints have been used in many different applications. Hence for this topic here we refer to [Baader and Snyder 2001] (Chapter 8 of this Handbook) for a detailed survey.

7.1. Ordering constraint solving

Apart from the applications to pruning the search space in automated theorem proving, ordering constraint solving is useful in many other contexts like proving termination of term rewrite systems or confluence of ordered term rewrite systems [Comon et al. 1998].

Regarding the former application, constraints provide powerful termination orderings \succ_c for term rewriting, defined: $s \succ_c t$ if $s\sigma \succ t\sigma$ for all ground σ . If \succ

is the recursive path ordering (RPO), such \succ_c subsume other path orderings like the ones of [Kapur, Narendran and Sivakumar 1985, Lescanne 1990] since all these path orderings coincide on ground terms (see [Dershowitz 1987]). For example, if s is $g(f(x), f(y))$ and t is $g(g(x, y), g(x, y))$, and $f \succ_{\mathcal{F}} g$ in the precedence, then $s \not\succ_{rpo} t$, but $s \succ_c t$. Ordering constraint solving is also applicable as an underlying technique in more general contexts like the *dependency pairs* method of [Arts and Giesl 1997].

As explained in Section 4.7 of this chapter, some applications of ordering constraints to ordered strategies in theorem proving gave rise to the distinction between fixed signature semantics (solutions are built over a given signature \mathcal{F}) and extended signature semantics (new symbols are allowed to appear in solutions) [Nieuwenhuis and Rubio 1992b].

The satisfiability problem for ordering constraints was first shown decidable for fixed signatures when \succ is a total LPO [Comon 1990] or a total RPO [Jouannaud and Okada 1991]. For extended signatures, decidability was shown for LPO in [Nieuwenhuis and Rubio 1995] and for RPO in [Nieuwenhuis 1993]. Regarding complexity, NP algorithms for LPO (fixed and extended signatures) and RPO (extended ones) were given in [Nieuwenhuis 1993]. Recently, an NP algorithm has been given as well for RPO under fixed signatures in [Narendran, Rusinowitch and Verma 1998]. For the AC-RPO ordering of [Rubio and Nieuwenhuis 1995], decidability was shown in [Comon, Nieuwenhuis and Rubio 1995]. NP-hardness of the satisfiability problem is known, even for one single inequation, for all these cases [Comon and Treinen 1994].

All these decision procedures use at some point the fact that a constraint C can be effectively expressed as an equivalent disjunction of expressions $s_1 > t_1 \wedge \dots \wedge s_n > t_n$, called *solved forms*, where for each i always at least one of s_i or t_i is a variable. Solved forms are obtained by repeatedly applying the definition of the ordering by rules like:

$$f(s_1, \dots, s_p) > t \implies s_1 \geq t \vee \dots \vee s_p \geq t$$

if t is a non-variable term whose topmost symbol is bigger in the precedence than f . Similar rules deal with the equality relations in the constraints, like $f(\dots) = g(\dots) \implies \perp$ if $f \neq g$, and $f(s_1, \dots, s_n) = f(t_1, \dots, t_n) \implies s_1 = t_1, \wedge \dots \wedge, s_n = t_n$.

Due to the transformations into disjunctive normal form, the number of solved forms for a given C may be exponential, even if all atoms in C are already inequalities between variables or if C consists of one single inequality. In algorithms like the ones of [Comon 1990] and [Nieuwenhuis and Rubio 1995], the computation of solved forms is only a first step that is followed by other exponential phases. This is not surprising, since this notion of solved form only involves a local analysis of the inequations considered independently. In fact any constraint $s > t$ can be expressed as the solved form $s > x \wedge x > t$, for some new variable x , which is equivalent w.r.t. satisfiability under extended signatures.

On the other hand, the NP algorithms of [Nieuwenhuis 1993] and [Narendran et al. 1998] are based on a first non-deterministic guess of a *simple system* for C , a particular constraint S of the form $s_n \#_n s_{n-1} \#_{n-1} \dots \#_1 s_0$, where

each $\#_i$ is either $=$ or $>$, and $\{s_n, \dots, s_1\}$ is the set of *all subterms* of C . Then, roughly, C is satisfiable if, and only if, some of its simple systems contains one of its own solved forms and entails C . For each simple system, this can be checked in polynomial time, but the number of simple systems to be considered is far too large for practical usefulness.

A new family of algorithms, for full RPO and both semantics, has been introduced recently in [Nieuwenhuis and Rivero 1999]. These algorithms are based on a new notion of solved form, where properties of orderings like transitivity and monotonicity are taken into account. They are simple and, since guessing is minimised, more efficient.

For the Knuth-Bendix ordering (KBO) the constraint satisfiability problem was proved decidable only recently [Korovin and Voronkov 2000a], and NP-complete in [Korovin and Voronkov 2000b]. Since theorem provers behave better on many problem classes with KBO than with path orderings like RPO, this result may become useful in practice.

8. Extensions

8.1. Paramodulation-based answer computation

Answer computation in some logic is at the heart of many applications of automated reasoning. Well-known simple examples of such mechanisms are Prolog's SLD-resolution, where the accumulated unifiers are kept as answers, or E-unification procedures for equational (or more general) theories E in which, given a goal $s = t$, answers σ are computed such that $E \models s\sigma = t\sigma$.

In [Gallier and Snyder 1989] general rules for E-unification are given. *Narrowing* was originally devised as an efficient E-unification procedure using a convergent (confluent and terminating) set of rewrite rules R for E [Fay 1979, Hullot 1980b]. Many extensions (to, e.g., conditional TRS's) and optimizations of narrowing have been proposed (see e.g. [Rety, Kirchner and Lescanne 1985, Bosco, Giovanetti and Moiso 1988, Hölldobler 1989, Nutt, Réty and Smolka 1989, Bockmayr, Krischer and Werner 1992, Bockmayr and Werner 1994]). Most completeness proofs of these narrowing strategies are based on lifting arguments applied to rewrite proofs, which has limitations when applied to unrestricted general clauses, more general simplification and redundancy notions, or with constrained clauses.

The techniques developed in this chapter can be extended into an alternative approach, based on the well-known fact that in refutation theorem proving, each refutation proof provides *one* answer, like in SLD-resolution. This has been done in [Nieuwenhuis 1995], where a proof technique is developed that uniformly covers E-unification-like methods and Prolog-like resolution strategies. By narrowing modulo equational theories like AC, compact representations of solutions, expressed by AC-equality constraints, can be obtained. Computing AC-unifiers is only needed at the end if one wants to "uncompress" such a constraint into its (doubly exponentially many) concrete substitutions. In [Lynch 1997] it is shown that superposition is

complete for answer computation with arbitrary selection rules (where also positive literals can be selected), thus properly extending SLD resolution to clauses with equality literals.

8.2. Paramodulation-based decidability and complexity results

The well-known close relationship between computation formalisms and deduction in some logic has been a starting point for a considerable amount of recent research in logic-based decidability and complexity analysis.

Regarding resolution-based results, for example in [Basin and Ganzinger 1996] saturatedness of sets S of clauses (without equality) with respect to different orderings implies membership in different complexity classes of the entailment problems $S \models C$ for ground C . And of course for the Datalog language of *flat* Horn clauses without equality there are a number of results from descriptive complexity theory; for example, that Datalog programs precisely capture the set of queries on a finite database decidable in finite time [Vardi 1982, Immerman 1986].

Regarding paramodulation-based decidability results, for the class of ground equations where some symbols are associative and commutative (AC) a finite confluent rewrite system can always be computed by superposition [Narendran and Rusinowitch 1991, Marché 1991], by which the word problem is decidable. In the same class, the unification problem is also decidable [Narendran and Rusinowitch 1993]. In [Marché 1996] paramodulation-based decidability results for word problems in ground presentations modulo several other theories extending AC are given, like abelian groups or commutative rings.

Concerning non-ground theories, superposition with simplification can be used as a decision procedure for the monadic class with equality [Bachmair, Ganzinger and Waldmann 1993b] (which is equivalent to a class of set constraints [Bachmair, Ganzinger and Waldmann 1993a]). Similar very recent results have been obtained for the guarded fragment [Ganzinger, Meyer and Veanes 1999, Ganzinger and de Nivelle 1999].

In [Waldmann 1999] it is shown that cancellative superposition decides the theory of divisible torsion-free abelian groups. The equational *shallow theories*, the ones axiomatized by equations where no variable occurs at depth more than one, are another fundamental class with decidable word and unification problems and even a decidable first-order theory [Comon, Haberstrau and Jouannaud 1994]. In [Nieuwenhuis 1998] it is shown that for sets of Horn clauses with equality saturated under basic paramodulation, the word and unifiability problems are in NP, and the number of minimal unifiers is simply exponential; this can be applied to shallow Horn clauses with equality. For certain Horn sets S saturated under basic superposition, the word and unifiability problems are still decidable and unification is finitary. Further results on the decidability of unification problems in Horn theories have been obtained by sorted superposition [Jacquemard, Meyer and Weidenbach 1998].

9. Perspectives

In this section some prominent research problems and future directions for research in this area are addressed.

Apart from adequate theoretical results as we have seen them in this chapter, building a state-of-the-art paramodulation-based theorem prover requires at least two more ingredients: good heuristics, and the necessary engineering skills to implement it all in an efficient way. Progress between theory and these other aspects is interacting in different ways.

On the one hand, new theoretical insights are replacing heuristics by more precise and effective techniques. For example, the completeness proof of basic paramodulation shows why no inferences below Skolem functions are needed, as conjectured by McCune [1990]. Regarding implementation techniques, ad-hoc algorithms for procedures like demodulation or subsumption are being replaced by efficient, re-usable, general-purpose indexing data structures for which the time and space requirements are well-known, see [Ramakrishnan, Sekar and Voronkov 2001] (Chapter 26 of this Handbook).

But, on the other hand, theory is also advancing in other directions, producing new ideas for which the development of implementation techniques and heuristics that make them applicable sometimes takes several years. For example, basic paramodulation was presented in 1992, but it was not applied in a state-of-the-art prover until four years later, when it was applied by McCune for finding his proof of the Robbins conjecture [McCune 1997b] by basic paramodulation modulo associativity and commutativity (AC).

Provers like Spass [Weidenbach 1997], based on (a still relatively small number of) such new theoretical insights, are now emerging and seem to be outperforming the “engineering-based” implementations of more standard calculi, in spite of still lacking more refined implementation techniques (see below).

McCune’s successful application of AC-paramodulation also illustrates the effectiveness—and the need—of building-in more and more knowledge about the problem domain (here, equality and the AC properties of some symbols) inside the general-purpose logics. Paramodulation with constraints seems to be an adequate paradigm for doing this in a clean way. It uses specialized techniques in the different constraint logics, supporting the reasoning process in the general-purpose logic. The interface between the two is through the variables: the constraints delimit the range of the quantifiers, and hence define the relevant instances of the expressions.

In the remainder of this section, some of the current theoretical and practical challenges concerning the construction of paramodulation-based provers are surveyed.

9.1. Basicness and redundancy

The basic restriction in paramodulation is easy to implement in most provers by marking *blocked* subterms, i.e., the point where the constraint starts. However, we

have seen that full simplification by demodulation is incomplete in combination with the basic strategy. An important challenge is to develop adequate redundancy notions for the basic strategy. Although some ideas are given in [Lynch and Scharff 1998], better results are needed for practice.

In the context of E-paramodulation with constraints, another interesting problem is how to apply a constrained equation $s \simeq t \mid T$ in a demodulation step without solving the E-unification problem in T . If the equation is small, and hence likely to be useful for demodulation, and the number of unifiers σ of T is small as well, it may pay off to keep some of the instantiated versions $s\sigma \simeq t\sigma$, along with the constrained equation, for use in demodulation. For large clauses this will probably not be useful.

9.2. Orderings

In all provers based on ordered strategies, the choice of the right ordering for a given problem turns out to be crucial. In many cases weaker (size- and weights-based) orderings like the Knuth-Bendix ordering behave well. In others, path orderings like LPO or RPO are better, although they depend heavily on the choice of the underlying precedence ordering on symbols. It is not clear at all how to choose orderings and precedences in practice. The prover can of course recognise familiar algebraic structures like groups or rings, and try orderings that normally behave well for each case, but is there no more general solution? For the case of E-paramodulation, these aspects are even less well-studied.

9.3. Constraint solving

As we have seen, for taking advantage of the constraints, algorithms for constraint satisfiability checking are required. Deciding the problem in general requires exponential time for path orderings like LPO or RPO. Is there any useful ordering for which deciding the satisfiability of (e.g., only conjunctive) constraints can be done in polynomial time? Or, if the answer is negative, for which orderings can we have better practical algorithms?

In practice one could use more efficient (sound, but incomplete) tests detecting most cases of unsatisfiable constraints: when a constraint T is unsatisfiable, the clause $C \mid T$ is redundant (in fact, it is a tautology) and can be removed. Are there any such tests?

In the context of a built-in theory E, equality constraint solving amounts to deciding E-unifiability problems. Although for many theories E a lot of work has been done on computing complete sets of unifiers, the decision problem has received less attention, see [Baader and Snyder 2001] (Chapter 8 of this Handbook). Are there any sound tests detecting most cases of unsatisfiability?

9.4. Indexing data structures

For many standard operations like many-to-one matching or unification indexing data structures exist that can be used in operations like inference computation, demodulation or subsumption, see [Ramakrishnan et al. 2001] (Chapter 26 of this Handbook). Such data structures are crucial in order to obtain a prover whose throughput remains stable while the number of clauses increases.

But for many operations no indexing data structures have been developed yet. For example, consider demodulation with equations that cannot be oriented *a priori* w.r.t. the ordering \succ , like the commutativity axiom. If such an equation $s \simeq t$ is found to be applicable to a term $s\sigma$, then after matching it has to be checked whether $s\sigma \succ t\sigma$, i.e., whether the corresponding rewrite step is indeed reductive. If it is not reductive, then the indexing data structure is asked to provide a new applicable equation, and so on. Of course it would be much better to have an indexing data structure that checks matching and ordering restrictions at the same time.

Apart from the AC case, indexing data structures for built-in E have received little attention. Especially for matching, at least for purely equational logic, they are really necessary. What are the perspectives for developing such data structures for other theories E?

9.5. More powerful redundancy notions

In the *Saturate* system [Nivela and Nieuwenhuis 1993, Ganzinger, Nieuwenhuis and Nivela 1999], a number of experiments with non-standard redundancy notions has been carried out. For example, *constrained rewriting* turns out to be powerful enough for deciding the confluence of ordered rewrite systems [Comon et al. 1998]. Other techniques based on forms of *contextual* and *clausal* rewriting can be used to produce rather complex saturatedness proofs for sets of clauses. In *Saturate*, the use of these methods is limited, since they are expensive (they involve search and ordering constraint solving) and *Saturate* is just an experimental Prolog implementation. However, from the experiments it is clear that such techniques importantly reduce the number of retained clauses.

Can such refined redundancy proof methods be implemented in a sufficiently efficient way to make them useful in real-world provers? It seems that their cost can be made independent of the size of the clause database of the prover (up to the size of the indexing data structures, but this is the case as well for simple redundancy methods like demodulation). Hence, they essentially slow down the prover in a (perhaps large) linear factor, but may produce an exponential reduction of the search space, thus being effective in hard problems.

9.6. *More global future research directions*

Up to this point, in this section we have focussed on concrete problems concerning the application of the theory explained in this chapter in actual provers. Longer-term challenges include the following two global areas.

One main area of interest concerns the integration of prover components: how to integrate dedicated procedures within general-purpose paramodulation-based provers (along the lines of Section 6), and how to integrate automated paramodulation-based provers in more general environments like proof assistants. Similarly, it has to be studied how to combine paramodulation-based provers with other automated reasoning paradigms.

A second major area of interest involves the application of the theory of paramodulation to other subfields of computer science like programming and complexity theory (along the lines of the results described in Sections 8.1 and 8.2), as well as more concrete applications like the analysis of security protocols [Weidenbach 1999].

Acknowledgments

We wish to thank the many people who helped us to improve (rewrite, polish, extend) this chapter, in particular Anatoli Degtyarev, Guillem Godoy, Jieh Hsiang, Chris Lynch, Michael Rusinowitch, and especially Andrei Voronkov.

Bibliography

- ARTS T. AND GIESL J. [1997], Automatically proving termination where simplification orderings fail, in 'TAPSOFT: 7th International Joint Conference on Theory and Practice of Software Development', LNCS 1214, Springer-Verlag, pp. 261–272.
- BAADER F. AND SNYDER W. [2001], Unification theory, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 8, pp. 445–532.
- BACHMAIR L. [1989], Proof normalization for resolution and paramodulation, in 'Third int. conf. on Rewriting Techniques and Applications (RTA)', LNCS 355, Springer-Verlag, Chapel Hill, NC, USA, pp. 15–28.
- BACHMAIR L. [1991], *Canonical equational proofs*, Birkhäuser, Boston, Mass.
- BACHMAIR L. AND DERSHOWITZ N. [1989], 'Completion for rewriting modulo a congruence', *Theoretical Computer Science* 2 and 3(67), 173–201.
- BACHMAIR L. AND DERSHOWITZ N. [1994], 'Equational inference, canonical proofs, and proof orderings', *J. of the Association for Computing Machinery* 41(2), 236–276.
- BACHMAIR L., DERSHOWITZ N. AND HSIANG J. [1986], Orderings for equational proofs, in 'First IEEE Symposium on Logic in Computer Science (LICS)', IEEE Computer Society Press, Cambridge, Massachusetts, USA, pp. 346–357.
- BACHMAIR L., DERSHOWITZ N. AND PLAISTED D. [1989], Completion Without Failure, in H. Aït-Kaci and M. Nivat, eds, 'Resolution of Equations in Algebraic Structures', Vol. 2: Rewriting Techniques, Academic Press, New York, chapter 1, pp. 1–30.
- BACHMAIR L. AND GANZINGER H. [1990], On restrictions of ordered paramodulation with simplification, in M. E. Stickel, ed., '10th International Conference on Automated Deduction (CADE)', LNAI 449, Springer-Verlag, Kaiserslautern, FRG, pp. 427–441.

- BACHMAIR L. AND GANZINGER H. [1991], Perfect model semantics for logic programs with equality, in K. Furukawa, ed., 'Logic Programming, Proceedings of the Eighth International Conference', The MIT Press, Paris, France, pp. 645–659.
- BACHMAIR L. AND GANZINGER H. [1994a], Associative-commutative superposition, in N. Dershowitz, ed., 'Proc. 5th International Workshop on Conditional Term Rewriting Systems', LNCS 968, Springer-Verlag, Jerusalem, pp. 155–167.
- BACHMAIR L. AND GANZINGER H. [1994b], 'Rewrite-based equational theorem proving with selection and simplification', *Journal of Logic and Computation* 4(3), 217–247.
- BACHMAIR L. AND GANZINGER H. [2001], Resolution theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19–99.
- BACHMAIR L., GANZINGER H., LYNCH C. AND SNYDER W. [1992], Basic paramodulation and superposition, in D. Kapur, ed., '11th International Conference on Automated Deduction (CADE)', LNAI 607, Springer-Verlag, Saratoga Springs, New York, USA, pp. 462–476.
- BACHMAIR L., GANZINGER H., LYNCH C. AND SNYDER W. [1995], 'Basic paramodulation', *Information and Computation* 121(2), 172–192.
- BACHMAIR L., GANZINGER H. AND WALDMANN U. [1993a], Set constraints are the monadic class, in 'Eighth Annual IEEE Symposium on Logic in Computer Science (LICS)', IEEE Computer Society Press, Montreal, Canada, pp. 75–83.
- BACHMAIR L., GANZINGER H. AND WALDMANN U. [1993b], Superposition with simplification as a decision procedure for the monadic class with equality, in '3rd Kurt Gödel Colloquium: Computational Logic and Proof Theory', LNCS 713, Springer-Verlag, pp. 83–96.
- BACHMAIR L. AND PLAISTED D. A. [1985], 'Termination orderings for associative-commutative rewriting systems', *Journal of Symbolic Computation* 1, 329–349.
- BASIN D. AND GANZINGER H. [1996], Complexity Analysis Based on Ordered Resolution, in 'Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS)', IEEE Computer Society Press, New Brunswick, New Jersey, USA, pp. 456–465.
- BEN-CHERIFA A. AND LESCANNE P. [1987], 'Termination of rewriting systems by polynomial interpretations and its implementation', *Science of Computer Programming* 9, 137–160.
- BOCKMAYR A., KRISCHER S. AND WERNER A. [1992], An optimal narrowing strategy for general canonical systems, in M. Rusinowitch and J.-L. Rémy, eds, 'The Third International Workshop on Conditional Term Rewriting Systems', LNCS 656, Springer-Verlag, Pont-à-Mousson, France.
- BOCKMAYR A. AND WERNER A. [1994], LSE narrowing for decreasing conditional term rewrite systems, in N. Dershowitz, ed., 'The fourth International Workshop on Conditional Term Rewriting Systems', LNCS 968, Jerusalem, pp. 167–190.
- BOFILL M., GODOY G., NIEUWENHUIS R. AND RUBIO A. [1999], Paramodulation with non-monotonic orderings, in '14th IEEE Symposium on Logic in Computer Science (LICS)', Trento, Italy, pp. 225–233.
- BOSCO P., GIOVANETTI E. AND MOISO C. [1988], 'Narrowing vs. sld-resolution', *Theoretical Computer Science* 2(59), 3–23.
- BOURELY C., CAFERRA R. AND PELTIER N. [1994], A method for building models automatically: Experiments with an extension of OTTER, in A. Bundy, ed., 'Proceedings of the 12th International Conference on Automated Deduction', Vol. 814 of LNAI, Springer, Berlin, pp. 72–86.
- BRAND D. [1975], 'Proving theorems with the modification method', *SIAM Journal on Computing* 4(4), 412–430.
- COMON H. [1990], 'Solving symbolic ordering constraints', *International Journal of Foundations of Computer Science* 1(4), 387–411.
- COMON H. [2001], Inductionless induction, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 14, pp. 913–962.
- COMON H., HABERSTRAU M. AND JOUANNAUD J.-P. [1994], 'Syntacticness, Cycle-Syntacticness and Shallow Theories', *Information and Computation* 111(1), 154–191.

- COMON H., NARENDRA P., NIEUWENHUIS R. AND RUSINOWITCH M. [1998], Decision problems in ordered rewriting, in '13th IEEE Symposium on Logic in Computer Science (LICS)', Indianapolis, USA, pp. 410–422.
- COMON H. AND NIEUWENHUIS R. [2000], 'Induction = I-axiomatization + first-order consistency', *Information and Computation*. To appear.
- COMON H., NIEUWENHUIS R. AND RUBIO A. [1995], Orderings, AC-Theories and Symbolic Constraint Solving, in '10th IEEE Symposium on Logic in Computer Science (LICS)', San Diego, USA, pp. 375–385.
- COMON H. AND TREINEN R. [1994], Ordering Constraints on Trees, in 'Proc. of Colloquium on Trees in Algebra and Programming (CAAP)', LNCS 787, Springer-Verlag, Edinburgh, Scotland, pp. 1–14.
- DEGTAREV A. [1979], The monotonic paramodulation strategy, in '5th All-Union Conference on Mathematical Logic', Novosibirsk. (In Russian).
- DEGTAREV A. AND VORONKOV A. [1986], 'Equality methods in machine theorem proving', *Cybernetics* **22**(3), 298–307.
- DEGTAREV A. AND VORONKOV A. [2001], Equality reasoning in sequent-based calculi, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 10, pp. 609–704.
- DERSHOWITZ N. [1982], 'Orderings for term-rewriting systems', *Theoretical Computer Science* **17**(3), 279–301.
- DERSHOWITZ N. [1987], 'Termination of rewriting', *Journal of Symbolic Computation* **3**, 69–116.
- DERSHOWITZ N. [1991], Canonical sets of Horn clauses, in J. L. Albert, B. Monien and M. R. Artelejo, eds, 'Proceedings of the Eighteenth International Colloquium on Automata, Languages and Programming (ICALP)', LNCS 510, Springer-Verlag, Madrid, Spain, pp. 267–278.
- DERSHOWITZ N. AND MANNA Z. [1979], 'Proving termination with multiset orderings', *Comm. of ACM* **22**(8).
- DERSHOWITZ N. AND PLAISTED D. [2001], Rewriting, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 9, pp. 533–608.
- DOMENJOU E. [1992], 'A technical note on AC-unification. the number of minimal unifiers of the equation $\alpha x_1 + \dots + \alpha x_p = \beta y_1 + \dots + \beta y_q$ ', *Journal of Automated Reasoning* **8**(1), 39–44.
- FAY M. [1979], First-order unification in an equational theory, in 'Proceedings of the Fourth Workshop on Automated Deduction', Austin, TX, pp. 161–167.
- GALLIER J. H. AND SNYDER W. [1989], 'Complete sets of transformations for general E-unification', *Theoretical Computer Science* **67**(2-3), 203–260.
- GANZINGER H. [1991], 'A completion procedure for conditional equations', *Journal of Symbolic Computation* **11**, 51–81.
- GANZINGER H. AND DE NIVELLE H. [1999], A superposition decision procedure for the guarded fragment with equality, in '14th IEEE Symposium on Logic in Computer Science (LICS)', Trento, Italy, pp. 295–305.
- GANZINGER H., MEYER C. AND VEANES M. [1999], The two-variable guarded fragment with transitive relations, in '14th IEEE Symposium on Logic in Computer Science (LICS)', Trento, Italy, pp. 24–34.
- GANZINGER H., NIEUWENHUIS R. AND NIVELA P. [1999], 'The Saturate System'. Software and documentation available at: <http://www.mpi-sb.mpg.de/SATURATE/Saturate.html>.
- GANZINGER H. AND WALDMANN U. [1996], Theorem proving in cancellative abelian monoids, in M. A. McRobbie and J. K. Slaney, eds, 'Thirteenth International Conference on Automated Deduction (CADE)', Vol. 1104 of *LNAI*, Springer, Berlin, pp. 388–402.
- GODOY G. AND NIEUWENHUIS R. [2000], Paramodulation with built-in abelian groups, in '15th IEEE Symposium on Logic in Computer Science (LICS)', Santa Barbara, USA.
- HÖLDOBLER S. [1989], *Foundations of equational logic programming*, LNCS 353, Springer-Verlag.

- HSIANG J. AND RUSINOWITCH M. [1987], On word problems in equational theories, in T. Ottmann, ed., 'Proc. 14th Int. Colloquium Automata, Languages and Programming', LNCS 267, Springer-Verlag, Berlin, Germany, pp. 54-71.
- HSIANG J. AND RUSINOWITCH M. [1991], 'Proving refutational completeness of theorem proving strategies: the transfinite semantic tree method', *Journal of the ACM* **38**(3), 559-587.
- HUET G. [1980], 'Confluent reductions: abstract properties and applications to term rewriting systems', *Journal of the ACM* **27**(4), 797-821.
- HULLOT J. [1980a], *Compilation de Formes Canoniques dans les Theories Equationnelles*, PhD thesis, Universite de Paris Sud, France.
- HULLOT J.-M. [1980b], Canonical forms and unification, in R. Kowalski, ed., 'Fifth International Conference on Automated Deduction (CADE)', LNCS 87, Springer-Verlag, Les Arcs, France, pp. 318-334.
- IMMERMAN N. [1986], 'Relational queries computable in polynomial time', *Information and Control* **68**(1-3), 86-104.
- JACQUEMARD F., MEYER C. AND WEIDENBACH C. [1998], Unification in extensions of shallow equational theories, in T. Nipkow, ed., '9th International Conference on Rewriting Techniques and Applications (RTA)', LNCS 1379, Springer, Tsukuba, Japan, pp. 76-90.
- JOUANNAUD J.-P. [1983], Confluent and coherent equational term rewriting systems: Applications to proofs in abstract data types, in 'Proc. 8th Colloquium on Trees in Algebra and Programming', LNCS 59, Springer-Verlag, pp. 269-283.
- JOUANNAUD J.-P. AND KIRCHNER H. [1986], 'Completion of a set of rules modulo a set of equations', *SIAM Journal of Computing* **15**, 1155-1194.
- JOUANNAUD J.-P. AND MARCHÉ C. [1992], 'Termination and completion modulo associativity, commutativity and identity', *Theoretical Computer Science* **104**, 29-51.
- JOUANNAUD J.-P. AND OKADA M. [1991], Satisfiability of systems of ordinal notations with the subterm property is decidable, in '18th International Colloquium Automata, Languages and Programming (ICALP)', LNCS 510, Springer-Verlag, Madrid, Spain, pp. 455-468.
- JOUANNAUD J.-P. AND WALDMANN B. [1986], Reductive conditional term rewriting systems, in 'Proc. Third IFIP Working Conference on Formal Description of Programming Concepts', Ebberup, Denmark.
- KAMIN S. AND LEVY J.-J. [1980], Two generalizations of the recursive path ordering. Unpublished note, Dept. of Computer Science, Univ. of Illinois, Urbana, IL.
- KAPLAN S. [1984], 'Conditional rewrite rules', *Theoretical Computer Science* **33**, 175-193.
- KAPUR D. AND NARENDRA P. [1992], Double exponential complexity of computing complete sets of AC-unifiers, in 'Seventh Annual IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, Santa Cruz, California, USA, pp. 11-21.
- KAPUR D., NARENDRA P. AND SIVAKUMAR G. [1985], A path ordering for proving termination for term rewriting systems, in 'Proc. of 10th Colloquium on Trees in Algebra and Programming', LNCS 185, Springer-Verlag, Germany, pp. 173-185.
- KAPUR D. AND SIVAKUMAR G. [1997], A total, ground path ordering for proving termination of ac-rewrite systems, in H. Comon, ed., '8th International Conference on Rewriting Techniques and Applications (RTA)', LNCS 1232, Springer-Verlag, Sitges, Spain, pp. 142-156.
- KAPUR D., SIVAKUMAR G. AND ZHANG H. [1990], A new method for proving termination of ac-rewrite systems, in 'Conf. Found. of Software Technology and Theoretical Computer Science', LNCS 472, Springer-Verlag, New Delhi, India, pp. 134-148.
- KIRCHNER C., KIRCHNER H. AND RUSINOWITCH M. [1990], 'Deduction with symbolic constraints', *Revue Française d'Intelligence Artificielle* **4**(3), 9-52.
- KNUTH D. AND BENDIX P. [1970], Simple word problems in universal algebras, in 'J. Leech, ed., Computational Problems in Abstract Algebra', Pergamon Press, Oxford, pp. 263-297.
- KOROVIN K. AND VORONKOV A. [2000a], A decision procedure for the existential theory of term algebras with the Knuth-Bendix ordering, in 'Proc. 15th Annual IEEE Symp. on Logic in Computer Science', Santa Barbara, California, pp. 291-302.

- KOROVIN K. AND VORONKOV A. [2000b], Knuth-bendix constraint solving is NP-complete, Preprint CSPP-8, Department of Computer Science, University of Manchester.
URL: <http://www.cs.man.ac.uk/preprints/index.html>
- KOUNALIS E. AND RUSINOWITCH M. [1991], 'On word problems in Horn theories', *Journal of Symbolic Computation* 11(1-2), 113-128.
- LANKFORD D. S. [1975], Canonical inference, Technical Report ATP-32, Dept. of Mathematics and Computer Science, Univ. of Texas, Austin, TX.
- LANKFORD D. S. AND BALLANTYNE A. M. [1977], Decision procedures for simple equational theories with commutative-associative axioms: Complete sets of commutative-associative reductions, Technical Report Memo ATP-39, Dept. of Mathematics and Computer Science, Univ. of Texas, Austin, TX.
- LESCANNE P. [1990], 'On the recursive decomposition ordering with lexicographical status and other related orderings', *Journal of Automated Reasoning* 6(1), 39-49.
- LOVELAND D. W. [1978], *Automated Theorem Proving: a Logical Basis*, 1 edn, North-Holland, Amsterdam.
- LYNCH C. [1997], 'Oriented equational logic programming is complete', *Journal of Symbolic Computation* 23(1), 23-46.
- LYNCH C. AND SCHARFF C. [1998], Basic completion with E-cycle simplification, in J. Calmet and J. Plaza, eds, 'Proceedings of the International Conference on Artificial Intelligence and Symbolic Computation (AISC-98)', LNAI 1476, Springer Verlag, pp. 209-221.
- LYNCH C. AND SNYDER W. [1993], Redundancy criteria for constrained completion, in C. Kirchner, ed., '5th International Conference on Rewriting Techniques and Applications (RTA)', LNCS 690, Springer-Verlag, Montreal, Canada, pp. 2-16.
- MARCHÉ C. [1991], On ground AC-completion, in R. V. Book, ed., '4th Int. Conf. Rewriting Techniques and Applications (RTA)', LNCS 488, Springer-Verlag, Como, Italy, pp. 411-422.
- MARCHÉ C. [1996], 'Normalized rewriting: An alternative to rewriting Modulo a set of equations', *Journal of Symbolic Computation* 21(3), 253-288.
- MARTIN U. AND NIPKOW T. [1990], Ordered rewriting and confluence, in M. E. Stickel, ed., '10th International Conference on Automated Deduction (CADE)', LNAI 449, Springer-Verlag, Kaiserslautern, FRG, pp. 366-380.
- MCCUNE W. [1990], Skolem functions and equality in automated deduction, in W. Dietterich, Tom; Swartout, ed., 'Proceedings of the 8th National Conference on Artificial Intelligence', MIT Press, Hynes Convention Centre?, pp. 246-251.
- MCCUNE W. [1994], OTTER 3.0 Reference Manual and Guide, Technical Report ANL-94/6, Argonne National Laboratory.
- MCCUNE W. [1997a], 33 basic test problems: A practical evaluation of some paramodulation strategies, in R. Veroff, ed., 'Automated Reasoning and its Applications: Essays in Honor of Larry Wos', MIT Press, pp. 71-114.
- MCCUNE W. [1997b], 'Solution of the Robbins problem', *Journal of Automated Reasoning* 19(3), 263-276.
- MCCUNE W. [1997c], Well behaved search and the Robbins problem, in H. Comon, ed., '8th International Conference on Rewriting Techniques and Applications (RTA)', LNCS 1232, Springer-Verlag, Sitges, Spain, pp. 1-7.
- NARENDRA P. AND RUSINOWITCH M. [1991], Any ground associative commutative theory has a finite canonical system, in '4th Int. Conf. Rewriting Techniques and Applications (RTA)', LNCS 488, Springer-Verlag, Como, Italy, pp. 423-434.
- NARENDRA P. AND RUSINOWITCH M. [1993], The Unifiability Problem in Ground AC Theories, in 'Eighth Annual IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, Montreal, Canada, pp. 364-370.
- NARENDRA P., RUSINOWITCH M. AND VERMA R. [1998], RPO constraint solving is in NP, in G. Gottlob, E. Grandjean and K. Seyr, eds, '12th Int. Conference of the European Association of Computer Science Logic (CSL)', LNCS 1584, Springer-Verlag, Brno, Czech Republic.

- NIEUWENHUIS R. [1993], 'Simple LPO constraint solving methods', *Information Processing Letters* **47**, 65–69.
- NIEUWENHUIS R. [1995], On Narrowing, Refutation Proofs and Constraints, in J. Hsiang, ed., '6th International Conference on Rewriting Techniques and Applications (RTA)', LNCS 914, Springer-Verlag, Kaiserslautern, Germany, pp. 56–70.
- NIEUWENHUIS R. [1998], 'Decidability and complexity analysis by basic paramodulation', *Information and Computation* **147**, 1–21.
- NIEUWENHUIS R. AND NIVELA P. [1991], 'Efficient deduction in equality horn logic by horn-completion', *Information Processing Letters* **39**(1), 1–6.
- NIEUWENHUIS R. AND OREJAS F. [1990], Clausal rewriting, in S. Kaplan and M. Okada, eds, 'Conditional and Typed Rewriting Systems, 2nd International Workshop', LNCS 516, Springer-Verlag, Montreal, Canada, pp. 246–258.
- NIEUWENHUIS R. AND RIVERO J. M. [1999], Solved forms for path ordering constraints, in P. Narendran and M. Rusinowitch, eds, 'Tenth International Conference on Rewriting Techniques and Applications (RTA)', LNCS 1631, Springer-Verlag, Trento, Italy, pp. 1–15.
- NIEUWENHUIS R. AND RUBIO A. [1992a], Basic superposition is complete, in B. Krieg-Brückner, ed., 'European Symposium on Programming', LNCS 582, Springer-Verlag, Rennes, France, pp. 371–390.
- NIEUWENHUIS R. AND RUBIO A. [1992b], Theorem proving with ordering constrained clauses, in D. Kapur, ed., '11th International Conference on Automated Deduction (CADE)', LNAI 607, Saratoga Springs, New York, pp. 477–491.
- NIEUWENHUIS R. AND RUBIO A. [1994], AC-Superposition with constraints: No AC-unifiers needed, in A. Bundy, ed., '12th International Conference on Automated Deduction (CADE)', LNAI 814, Springer-Verlag, Nancy, France, pp. 545–559.
- NIEUWENHUIS R. AND RUBIO A. [1995], 'Theorem Proving with Ordering and Equality Constrained Clauses', *Journal of Symbolic Computation* **19**(4), 321–351.
- NIEUWENHUIS R. AND RUBIO A. [1997], 'Paramodulation with Built-in AC-Theories and Symbolic Constraints', *Journal of Symbolic Computation* **23**(1), 1–21.
- NIVELA P. AND NIEUWENHUIS R. [1993], Practical results on the saturation of full first-order clauses: Experiments with the saturate system. (system description), in C. Kirchner, ed., '5th International Conference on Rewriting Techniques and Applications (RTA)', LNCS 690, Springer-Verlag, Montreal, Canada, pp. 436–440.
- NUTT W., RÉTY P. AND SMOLKA G. [1989], 'Basic narrowing revisited', *Journal of Symbolic Computation* **7**, 295–317.
- PAIS J. AND PETERSON G. [1991], 'Using Forcing to Prove Completeness of Resolution and Paramodulation', *Journal of Symbolic Computation* **11**(1), 3–19.
- PAUL E. [1992], 'A general refutational completeness result for an inference procedure based on associative-commutative unification', *Journal of Symbolic Computation* **14**(6), 577–618.
- PETERSON G. E. [1983], 'A technique for establishing completeness results in theorem proving with equality', *SIAM J. on Computing* **12**(1), 82–100.
- PETERSON G. E. [1990], Complete sets of reductions with constraints, in M. E. Stickel, ed., '10th International Conference on Automated Deduction (CADE)', LNAI 449, Springer-Verlag, Kaiserslautern, FRG, pp. 381–395.
- PETERSON G. AND STICKEL M. [1981], 'Complete sets of reductions for some equational theories', *Journal Assoc. Comput. Mach.* **28**(2), 233–264.
- PLOTKIN G. [1972], 'Building in equational theories', *Machine Intelligence* **7**, 73–90.
- PRZYMUSIŃSKA H. AND PRZYMUSIŃSKI T. [1990], 'Weakly Stratified Logic Programs', *Fundamenta Informaticae* **XIII**, 51–65.
- PRZYMUSIŃSKI T. [1988], On the declarative semantics of deductive databases and logic programs, in 'Foundations of deductive databases and logic programming', Morgan Kaufmann, Los Altos, CA., pp. 193–216.

- RAMAKRISHNAN I., SEKAR R. AND VORONKOV A. [2001], Term indexing, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 26, pp. 1853–1964.
- RETY P., KIRCHNER C. AND LESCANNE P. [1985], NARROWER: a new algorithm for unification and its application to logic programming, in J.-P. Jouannaud, ed., '1st International Conference Rewriting Techniques and Applications (RTA)', LNCS 202, Springer-Verlag, Dijon, France, pp. 141–157.
- ROBINSON G. A. AND WOS L. T. [1969], 'Paramodulation and theorem-proving in first order theories with equality', *Machine Intelligence* 4, 135–150.
- ROBINSON J. A. [1965], 'A machine-oriented logic based on the resolution principle', *Journal of the ACM* 12(1), 23–41.
- RUBIO A. [1994], 'Automated deduction with ordering and equality constrained clauses', PhD. Thesis, Technical University of Catalonia, Barcelona, Spain.
- RUBIO A. [1995], Extension Orderings, in '22nd International Colloquium on Automata, Languages and Programming (ICALP)', LNCS 944, Springer-Verlag, Szeged, Hungary, pp. 511–522.
- RUBIO A. [1996], Theorem proving modulo associativity, in '9th Int. Conference of the European Association for Computer Science Logic (CSL)', LNCS 1092, Springer-Verlag, Paderborn, Germany, pp. 452–467.
- RUBIO A. [1999], A fully syntactic AC-RPO, in P. Narendran and M. Rusinowitch, eds, 'Tenth International Conference on Rewriting Techniques and Applications (RTA)', LNCS 1631, Springer-Verlag, Trento, Italy, pp. 133–147.
- RUBIO A. AND NIEUWENHUIS R. [1995], 'A total AC-compatible ordering based on RPO', *Theoretical Computer Science* 142(2), 209–227.
- RUSINOWITCH M. AND VIGNERON L. [1995], 'Automated deduction with associative commutative operators', *J. of Applicable Algebra in Engineering, Communication and Computation* 6(1), 23–56.
- SLAGLE J. R. [1974], 'Automated theorem-proving for theories with simplifiers, commutativity, and associativity', *Journal of the ACM* 21(4), 622–642.
- SNYDER W. AND LYNCH C. [1991], Goal directed strategies for paramodulation, in R. V. Book, ed., '4th Int. Conf. Rewriting Techniques and Applications (RTA)', LNCS 488, Springer-Verlag, Como, Italy, pp. 150–161.
- STUBER J. [1998a], 'Superposition theorem proving for abelian groups represented as integer modules', *Theoretical Computer Science* 208(1–2), 149–177.
- STUBER J. [1998b], Superposition theorem proving for commutative rings, in W. Bibel and P. H. Schmitt, eds, 'Automated Deduction - A Basis for Applications. Volume III. Applications', Kluwer, Dordrecht, The Netherlands, chapter 2, pp. 31–55.
- SUTCLIFFE G. AND SUTTNER C. B. [1998], The CADE-14 ATP system competition, Technical Report JCU-CS-98/01, Department of Computer Science, James Cook University.
URL: <http://www.cs.jcu.edu.au/ftp/pub/techreports/98-01.ps.gz>
- VARDI M. Y. [1982], The complexity of relational query languages (extended abstract), in 'Proceedings 14th Annual ACM Symp. on Theory of Computing, STOC'82, San Francisco, CA, USA, 5–7 May 1982', ACM Press, New York, pp. 137–146.
- VIGNERON L. [1994], Associative Commutative Deduction with constraints, in A. Bundy, ed., '12th International Conference on Automated Deduction (CADE)', LNAI 814, Springer-Verlag, Nancy, France, pp. 530–544.
- VIGNERON L. [1996], Positive deduction modulo regular theories, in '9th Int. Conference of the European Association for Computer Science Logic (CSL)', LNCS 1092, Springer-Verlag, Paderborn, Germany, pp. 468–486.
- WALDMANN U. [1998], Superposition for divisible torsion-free abelian monoids, in 'Proceedings of the Fifteenth International Conference on Automated Deduction (CADE-98)', Vol. 1421 of LNAI, Springer, Berlin, pp. 144–159.

- WALDMANN U. [1999], Cancellative superposition decides the theory of divisible torsion-free abelian groups, in 'Logic Programming and Automated Reasoning, Int. Conf.', LNAI 1705, Springer-Verlag, Tbilisi, Georgia, pp. 131–147.
- WEIDENBACH C. [1997], 'SPASS—version 0.49', *Journal of Automated Reasoning* 18(2), 247–252.
- WEIDENBACH C. [1999], Towards an automatic analysis of security protocols in first-order logic, in H. Ganzinger, ed., 'Proceedings of the 16th International Conference on Automated Deduction (CADE-16)', Vol. 1632 of *LNAI*, Springer-Verlag, Berlin, pp. 314–328.
- WERTZ U. [1992], First-order theorem proving modulo equations, Technical Report MPI-I-92-216, Max-Planck-Institut für Informatik, Saarbrücken.
- Wos L. [1988], *Automated Reasoning: 33 Basic Research Problems*, Prentice-Hall, Englewood Cliffs.
- Wos L. [1996], *The Automation of Reasoning: An Experimenter's Notebook with OTTER Tutorial*, Academic Press.
- Wos L., ROBINSON G. A., CARSON D. F. AND SHALLA L. [1967], 'The concept of demodulation in theorem proving', *Journal of the ACM* 14(4), 698–709.
- ZHANG H. [1988], Reduction, Superposition, and Induction: Automated Reasoning in an Equational Logic, PhD thesis, Rensselaer Polytechnic Institute.

Index

Symbols

$=$	383
$\gamma_{\mathcal{F}}$	383
$=_E$	382
$=_R$	382
$D \mid_p$	374
E^*	382
R^*	382
R_C	387
$S \prec^C$	401
\square	384
\mathcal{G}	386
\mathcal{H}	391
\mathcal{I}	394
\mathcal{S}	397
\leftarrow	382
\mapsto^*_E	382
\mapsto^*_R	382
\nearrow	383
\nearrow_{rpo}	383
γ_{lex}	383
γ_c	386, 387
γ_{mul}	383
γ_{rpo}	383
γ_{lex}^{rpo}	383
\rightarrow	382
\rightarrow^*	382
\rightarrow^*_E	375
\rightarrow^+_+	382
$t[s]_p$	381
$t \mid_p$	381
$vars(t)$	381
\mathcal{I}_{AC}	423

A

AC-superposition	424
AC-top-superposition	424
AC-unification	380
answer computation	427
antecedent	384
Argonne group	374

B

basic	
paramodulation	378, 417
strategy	378
superposition	378
basicness	
restriction	380
blocked positions	417

built-in equational theories	421
built-in theories	379

C

Church-Rosser property	382
clausal rewriting	376
clause	384
constrained	385, 414
constrained clause	
equality	378
irreducibility	378
clauses	
general	394
Horn	386
closure substitution	378
compatibility	
AC-compatibility	383
E-compatibility	383
completion	
E-completion	379
Knuth-Bendix	375
modulo AC	379
unfailing	375
conditional equation	376
conditional rewriting	376
confluence	375, 382
ground	413
congruence	382
axioms	373
congruence axioms	373
consistency proving	377
constrained clause	378, 385, 414
equality	378
general	420
constrained empty clause	385
constrained formulae	378
constraint	378, 385
decidability of	379
equality	385, 425
equality constrained clause	378
extended signature	426
fixed signature	426
global	393
inheritance strategies	379
local	393
ordering	385, 425
satisfiable	385
solution	385
solving	379, 425
complexity	426

decidability	426
symbolic	425
tautology	385
without inheritance	393
constraint inheritance	
with built-in E	424
convergence	382
critical pair	375
criteria	376

D

Datalog	428
decision procedures	412
demodulation	377
derivation	400, 402

E

E-unification	380
E-unifier	380
empty clause	384
EQP	374
equality constraint	
inheritance	414
equality factoring	395
equality resolution	386, 395
equation	382
conditional	376
extended E-rewriting	379

F

factoring	373
fair derivations	404
fairness	404
ground	408
in practice	404, 409
non-ground	406
follows from	384
forcing	376
functional reflexivity axioms	374

G

ground	
substitution	381
term	381
ground confluence	413
ground fair	408

H

Herbrand interpretation	384
-------------------------------	-----

I

inductive proofs	377
inference	385
rule	385

complete	385
correct	385
system	385
AC-clauses \mathcal{I}_{AC}	423
general clauses \mathcal{I}	394
general clauses \mathcal{M}	397
general clauses \mathcal{S}	397
ground Horn clauses \mathcal{G}	386
Horn clauses \mathcal{H}	391
with selection \mathcal{S}	397
instance	381
interpretation	384
equality Herbrand	384
irreducibility	382
irreducible substitution	420

K

Knuth-Bendix completion	375
-------------------------------	-----

L

Leibniz	374
lexicographic path ordering (LPO)	384
lifting	394, 406
argument	406
local confluence	382
logical consequence	384
LPO	384

M

matching	381
merging paramodulation	397
mgu	373, 381
minimal Herbrand model	398
model	
minimal Herbrand	398
perfect	398
model generation	376, 389
for general clauses	396
method	386
monotonic	382
monotonicity axioms	374
most general unifier	373
multiset	382

N

narrowing	378, 427
normal form	375, 382

O

ordered factoring	397
ordered paramodulation	374, 376
ordered resolution	390
ordering	383
compatible	383

E-compatible	383
reduction	375, 383
simplification	383
term	374
well-founded	383
ordering constraint	
inheritance	417
orderings	
A-compatible	422
AC-compatible	422
ACD-compatible	423
ACU-compatible	423
C-compatible	422
combination of	413
E-compatible	422
I-compatible	423
rewrite	383
Otter	374

P

paramodulation	373, 374
-based complexity analysis	428
-based decidability analysis	428
basic	417
modulo AC	421, 423
modulo E	421
ordered	374, 376
path ordering	
lexicographic (LPO)	384
recursive (RPO)	384
path orderings	384
perfect model	398
persistent	402
position	381
positive strategies	397
positive unit strategies	397
predicates	
non-equality	390
proof orderings	376

R

recursive path ordering (RPO)	384
reduction ordering	383
redundancy	376, 402
abstract	376
backward	399
forward	399
practical methods	402
redundant	401
clause	376, 402
inference	376, 401
refutation complete	373
relation	382

Church-Rosser	382
confluent	382
inverse	382
locally confluent	382
monotonic	382
normal form	382
reflexive-transitive closure	382
terminating	382
transitive closure	382
well-founded	382
resolution	373
binary	373
rewrite	
rule	382
system	382
rewrite ordering	383
rewrite system	382
Church-Rosser	382
confluent	382
locally confluent	382
terminating	382
rewriting	375, 382
clausal	376
conditional	376
extended E-	379
modulo E	379
ordered	375
Robbins conjecture	374
Robbins problem	380
Robinson	373
RPO	384

S

Saturate	377
saturated set	400
saturated sets	377
combination of	413
finite	377
saturation	376, 377, 400, 401
for constrained clauses	418
non-ground	405
procedures	377, 401
selection	397
set-of-support	377
signature	381
extended	413, 426
fixed	413, 426
simplification ordering	383
Skolem symbols	413
solution of a constraint	385
solved forms	426
Spass	377
stability	

under substitutions	383
strategies	
selection	397
strict superposition	376
substitution	375, 381
application	381
ground	381
irreducible	415
subsumption	377, 381
subterm	381
property	383
succedent	384
superposition	375, 386
basic	378
left	386, 395
right	386, 395
strict	376

T

term	381
maximal	386
position	381
strictly maximal	386
term ordering	374
term rewrite system	382
termination	382
theories	
built-in	379
totality	383
transfinite semantic trees	376

U

unfailing completion	375
unification	381
AC	380
unifier	381

W

well-constrained sets	415
word problem	375

Unification Theory

Franz Baader

Wayne Snyder

SECOND READERS: Paliath Narendran, Manfred Schmidt-Schauss, and
Klaus Schulz.

Contents

1	Introduction	447
1.1	What is unification?	447
1.2	History and applications	448
1.3	Approach	450
2	Syntactic unification	450
2.1	Definitions	451
2.2	Unification of terms	452
2.3	Unification of term <i>dags</i>	459
3	Equational unification	469
3.1	Basic notions	469
3.2	New issues	473
3.3	Reformulations	475
3.4	Survey of results for specific theories	482
4	Syntactic methods for <i>E</i> -unification	488
4.1	<i>E</i> -unification in arbitrary theories	488
4.2	Restrictions on <i>E</i> -unification in arbitrary theories	495
4.3	Narrowing	495
4.4	Strategies and refinements of basic narrowing	499
5	Semantic approaches to <i>E</i> -unification	503
5.1	Unification modulo <i>ACU</i> , <i>ACUI</i> , and <i>AG</i> : an example	504
5.2	The class of commutative/monoidal theories	508
5.3	The corresponding semiring	510
5.4	Results on unification in commutative theories	511
6	Combination of unification algorithms	513
6.1	A general combination method	514
6.2	Proving correctness of the combination method	517
7	Further topics	519
	Bibliography	521
	Index	531

HANDBOOK OF AUTOMATED REASONING

Edited by Alan Robinson and Andrei Voronkov

© 2001 Elsevier Science Publishers B.V. All rights reserved

1. Introduction

Unification is a fundamental process upon which many methods for automated deduction are based. Unification theory abstracts from the specific applications of this process: it provides formal definitions for important notions like instantiation, most general unifier, etc., investigates properties of these notions, and provides and analyzes unification algorithms that can be used in different contexts. In this introductory section, we will first present the concept of unification in an informal way, then make some historical remarks on where unification was originally introduced, and finally explain our approach to writing this chapter.

1.1. What is unification?

Very generally speaking, unification tries to identify two symbolic expressions by replacing certain sub-expressions (variables) by other expressions. To be more concrete, one usually considers *terms* that are built from function symbols (say f , a , and b , where f is binary and a, b are nullary) and variable symbols (say x and y). The *unification problem* for the terms $s = f(a, x)$ and $t = f(y, b)$ is concerned with the following question: is it possible to replace the variables x, y in s and t by terms such that the two terms obtained this way are (syntactically) equal. In this example, if we substitute b for x and a for y , we obtain the *unified term* $f(a, b)$. This substitution is denoted as $\sigma := \{x \mapsto b, y \mapsto a\}$, and its application to terms is written suffix, i.e., $s\sigma = f(a, b) = t\sigma$. Note that different occurrences of the same variable in a unification problem must always be replaced by the same term. For this reason, the terms $s' = f(a, x)$ and $t' = f(x, b)$ cannot be unified since this would require the occurrence of x in s' to be replaced by b , and the occurrence of x in t' to be replaced by the different constant a .

In most applications of unification, one is not just interested in the *decision problem* for unification, which simply asks for a “yes” or “no” answer to the question of whether two terms s and t are unifiable. If they are unifiable, one would like to construct a solution, i.e., a substitution that identifies s and t . Such a substitution is called a *unifier* of s and t . In general, a unification problem may have infinitely many solutions; e.g., $f(x, y)$ and $f(y, x)$ can be unified by replacing x and y by the same term s (and there are infinitely many terms available). Fortunately, the applications of unification in automated deduction do not require the computation of all unifiers. It is sufficient to consider the so-called *most general unifier*, i.e., a unifier such that every other unifier can be obtained by *instantiation*. In the above example, $\sigma := \{x \mapsto y\}$ is such a most general unifier since for all terms s we have $\{x \mapsto s, y \mapsto s\} = \sigma\{y \mapsto s\}$. A unification algorithm should thus not only decide solvability of a given unification problem: if the problem is solvable, it should also compute a most general unifier. As we will show, there exist very efficient algorithms for this purpose.

Unification as described until now is called *syntactic* unification of *first-order* terms. “Syntactic” means that the terms must be made syntactically equal, whereas

“first-order” expresses the fact that we do not allow for higher-order variables, i.e., variables for functions. For example, the terms $f(x, a)$ and $g(a, x)$ obviously cannot be made syntactically equal by first-order unification. However, $f(x, a)$ and $G(a, x)$ can be made equal by *higher-order unification* if G is a (higher-order) variable, which may be replaced by f . We will not consider higher-order unification in more detail since it is treated in [Dowek 2001] (Chapter 16 of this Handbook). However, *equational unification*—as opposed to syntactic unification—of first-order terms will be one of the most important topics of this chapter. Instead of requiring that the terms are made syntactically equal, equational unification is concerned with making terms equivalent with respect to a congruence induced by certain equational axioms E . In this case, one talks about E -unification or unification modulo E . For example, if $E = \{f(a, a) \approx g(a, a)\}$, then the terms $f(x, a)$ and $g(a, x)$, which are not (syntactically) unifiable, are E -unifiable: for the substitution $\sigma := \{x \mapsto a\}$, we have $f(x, a)\sigma = f(a, a) =_E g(a, a) = g(a, x)\sigma$, where $=_E$ denotes the equational theory induced by E . For equational unification, things are not as nice as for syntactic unification. In fact, depending on the theory E in question, E -unifiability may be undecidable, and even if it is decidable, solvable unification problems need not have a most general E -unifier. Research on equational unification is, on the one hand, interested in classifying equational theories of interest according to their behavior in this respect. On the other hand, it develops general approaches and algorithms that apply to whole classes of theories.

1.2. History and applications

The name “unification” and the first formal investigation of this notion is due to J.A. Robinson [1965], who introduced unification as the basic operation of his resolution principle, showed that unifiable terms have a most general unifier, and described an algorithm for computing this unifier. In the propositional case, the resolution principle can be described as follows, see also [Bachmair and Ganzinger 2001] (Chapter 2 of this Handbook). Assume that clauses $C \vee p$ and $C' \vee \neg p$ have already been derived (where C, C' are sub-clauses and p is a propositional variable). Then one can also deduce $C \vee C'$. In the first-order case, the clauses one starts with may contain variables. Herbrand’s famous theorem implies that finitely many ground instances (i.e., instances obtained by substituting all variables by terms without variables) are sufficient to show unsatisfiability of a given unsatisfiable set of clauses by propositional reasoning (e.g., propositional resolution). The problem is, however, to find the appropriate instantiations. Early theorem provers approached this problem by a breadth-first enumeration of all possible ground instantiations, which led to an immediate combinatorial explosion [Robinson 1963]. Theorem provers based on the resolution principle need not search blindly for the right instantiations: they can compute them via syntactic unification. For example, assume the clauses $C \vee P(s)$ and $C' \vee \neg P(t)$ are given. Obviously, the resolution rule applies to ground instances of these clauses iff in these instances the predicate P contains the same term, i.e., iff the substitution used in the instantiation process is a (syntactic) unifier of s and t .

Instead of using all ground unifiers for instantiation, Robinson proposed to lift the resolution principle to terms with variables, and apply only the most general unifier σ of s and t . In the example, this yields the resolvent $(C \vee C')\sigma$. The completeness proof for propositional resolution can be lifted to non-ground resolution by using the fact that every ground unifier of s, t is an instance of the most general unifier. In fact, the notion “most general unifier” was defined in this way just to make this lifting possible.

Similar ideas for determining appropriate instantiations have been proposed prior to Robinson by Post, Herbrand [1930a, 1930b, 1967, 1971] (in the investigation of his property A), Prawitz [1960], and Guard [1964, 1969]. However, in this previous work, the notions “unification” and “most general unifier” are not singled out as interesting concepts of their own (they don’t even receive a name). Prawitz only considers the function-free case (in which unification is rather trivial), and Herbrand also first presents his approach for this restricted case. The description by Herbrand of the unification algorithm for the general case (which appears to be the first published account of such an algorithm, and which is similar to the transformation-based algorithm by Martelli and Montanari [1982]) is rather informal, and there is no proof of correctness.¹

The notions “unification” and “most general unifier” were independently reinvented by Knuth and Bendix [1970] as a tool for testing term rewriting systems for local confluence by computing critical pairs. Again, the definition of the most general unifier makes sure that every critical situation is an instance of a critical pair, and thus it is sufficient to test the critical pairs for confluence, see [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook). *Equational unification* was introduced both in resolution-based theorem proving and in term rewriting as a means for treating certain troublesome equational axioms (like associativity and commutativity) in a special manner. In automated theorem proving, it quickly became apparent that the equality relation requires a special treatment (see [Degtyarev and Voronkov 2001a, Nieuwenhuis and Rubio 2001], Chapters 10 and 7 of this Handbook), since a simple integration of axioms that describe the properties of equality (in principle, being a congruence relation) often leads to an unacceptable increase in the search space. Whereas the first approaches providing such a special treatment of equality replaced only the axiomatization of equality by special inference rules, Plotkin [1972] proposed to go one step further. In his approach, also certain axioms that use equality (like $f(x, y) \approx f(y, x)$ and $f(f(x, y), z) \approx f(x, f(y, z))$) can be built into the inference rule (namely resolution). This is achieved by replacing the use of syntactic unification in the resolution step by equational unification, i.e., unification modulo the equational theory induced by the axioms to be built in.

In term rewriting, axioms like commutativity (i.e., $f(x, y) \approx f(y, x)$) cannot be oriented into terminating rewrite rules. One way of solving this problem is to take such non-orientable identities completely out of the rewrite process, and perform

¹Strictly speaking, Herbrand’s unification algorithm is not an algorithm for simple syntactic unification, but an algorithm for unification with so-called linear constant restrictions (see section 3.3.2). This is due to the fact that he does not Skolemize his formulae, and thus he has both universal and existential quantifiers in the quantifier prefix.

rewriting with respect to the remaining (orientable) rules modulo the unoriented ones. In this setting, critical pairs must now be computed by equational unification, i.e., modulo the unoriented identities, see, e.g., [Peterson and Stickel 1981, Jouannaud and Kirchner 1986] and [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook).

1.3. Approach

This chapter is not intended to give a complete coverage of all the results obtained in unification theory (see the overview articles [Jouannaud and Kirchner 1991, Baader and Siekmann 1994] for this purpose). Instead we try to cover a number of significant topics in more detail. This should give a feeling for unification research and its methodology, provide the most important references, and enable the reader to study recent research papers on the topic.

Notational and typographic conventions

We will try to keep as close as possible to the typographic conventions introduced by Dershowitz and Jouannaud [1991], which they also used in their survey article on rewrite systems [Dershowitz and Jouannaud 1990]. In particular, substitutions are written in suffix notation (i.e., $s\sigma$ instead of $\sigma(s)$), and consequently composition of substitution should be read from left to right (i.e., $\sigma\tau$ means: first apply σ and then τ).

Equational axioms (written $s \approx t$) that define equational theories will be called “identities,” whereas unification problems consist of “equations” (written $s =^? t$ for syntactic unification and $s =_E^? t$ for unification modulo E). Thus, identities must hold, whereas equations must be solved.

2. Syntactic unification

As mentioned earlier, syntactic unification of first-order terms was introduced by Post and Herbrand in the early part of this century. Various researchers have studied the problem further [Champeaux 1986, Corbin and Bidoit 1983, Huet 1976, Martelli and Montanari 1982, Paterson and Wegman 1978, Robinson 1971, Venturini-Zilli 1975] and, among other results, it was shown that linear time algorithms for unification exist [Martelli and Montanari 1976, Paterson and Wegman 1978]. The corresponding lower complexity bound was shown by Dwork, Kanellakis and Mitchell [1984]: the unification problem is log-space complete for P , the class of polynomial-time solvable problems. In particular, this implies that it is very unlikely that an efficient parallel unification algorithm exists.

In this section we review the major approaches to syntactic unification.

2.1. Definitions

A *signature* is a (finite or countably infinite) set of function symbols \mathcal{F} . We assume the reader is familiar with the term algebra $\mathcal{T}(\mathcal{F}, \mathcal{V})$ generated by a signature function symbols \mathcal{F} and a (countably) infinite set of variables \mathcal{V} ; we shall call these \mathcal{F} -terms, or simply *terms* when \mathcal{F} is unimportant, and denote them by the letters l, r, s, t, u , and v . Variables will be denoted by w, x, y , and z . The set of variables occurring in a term t will be denoted by $\text{Vars}(t)$, and this will be extended to sets of variables, equations, and sets of equations.

A substitution is a mapping from variables to terms which is almost everywhere equal to the identity, and will generally be represented by σ, θ, η , or ρ . The identity substitution is represented by Id . The application of a substitution σ to a term t , denoted $t\sigma$, is defined by induction on the structure of terms:

$$t\sigma := \begin{cases} x\sigma & \text{if } t = x, \\ f(t_1\sigma, \dots, t_n\sigma) & \text{if } t = f(t_1, \dots, t_n). \end{cases}$$

In the second case of this definition, $n = 0$ is allowed: in this case, f is a constant symbol and $f\sigma = f$. Substitutions can also be applied to sets of terms, equations, and sets of equations, in the obvious fashion.

For a substitution σ , the *domain* is the set of variables

$$\text{Dom}(\sigma) := \{x \mid x\sigma \neq x\},$$

the *range* is the set of terms

$$\text{Ran}(\sigma) := \bigcup_{x \in \text{Dom}(\sigma)} \{x\sigma\},$$

and the set of variables occurring in the range is $\mathcal{VRan}(\sigma) := \text{Vars}(\text{Ran}(\sigma))$. A substitution can be represented explicitly as a function by a set of bindings of variables in its domain, e.g.,

$$\{x_1 \mapsto s_1, \dots, x_n \mapsto s_n\}.$$

The *restriction* of a substitution θ to a set of variables X , denoted by $\theta|_X$, is the substitution which is equal to the identity everywhere except over $X \cap \text{Dom}(\theta)$, where it is equal to θ . *Composition* of two substitutions is written $\sigma\theta$, and is defined by

$$t\sigma\theta = (t\sigma)\theta.$$

An algorithm for constructing the composition $\sigma\theta$ of two substitutions represented as sets of bindings is as follows:

1. Apply θ to every term in $\text{Ran}(\sigma)$ to obtain σ_1 ;
2. Remove from θ any binding $x \mapsto t$, where $x \in \text{Dom}(\sigma)$, to obtain θ_1 ;
3. Remove from σ_1 any trivial binding $x \mapsto x$, to obtain σ_2 ; and

4. Take the union of the two sets of bindings σ_2 and θ_1 .

It is also useful to be able to represent substitutions in their *triangular form* as a sequential list of bindings, e.g.,

$$[x_1 \mapsto t_1; x_2 \mapsto t_2; \dots; x_n \mapsto t_n],$$

which represents the composition of n substitutions each consisting of a single binding:

$$\{x_1 \mapsto t_1\} \{x_2 \mapsto t_2\} \dots \{x_n \mapsto t_n\}.$$

A substitution is *idempotent* if $\sigma\sigma = \sigma$; it is easy to show that this is true iff $\text{Dom}(\sigma) \cap \text{VRan}(\sigma) = \emptyset$.

A *variable renaming* substitution is defined as a substitution σ such that $\text{Dom}(\sigma) = \text{Ran}(\sigma)$. (For example, $\{x \mapsto y, y \mapsto z, z \mapsto x\}$ is a variable renaming, whereas $\{x \mapsto y\}$ and $\{y \mapsto z, x \mapsto z\}$ are not.) For any such variable renaming $\rho = \{x_1 \mapsto y_1, \dots, x_n \mapsto y_n\}$, we denote its inverse $\{y_1 \mapsto x_1, \dots, y_n \mapsto x_n\}$ by ρ^{-1} .

Two substitutions are equal, denoted $\sigma = \theta$, if $x\sigma = x\theta$ for every variable x . We say that σ is *more general than* θ , denoted $\sigma \leq \theta$, if there exists an η such that $\theta = \sigma\eta$. The relation \leq is called the *instantiation quasi-ordering*. The corresponding equivalence relation (i.e., $\leq \cap \geq$) is denoted by \doteq ; it can be shown [Lassez, Maher and Mariott 1987] that $\sigma \doteq \theta$ iff there exists some variable renaming ρ such that $\sigma = \theta\rho$.

2.1. DEFINITION. A substitution σ is a *unifier* of two terms s and t if $s\sigma = t\sigma$; it is a *most general unifier* (or *mgu* for short), if for every unifier θ of s and t , $\sigma \leq \theta$. A *unification problem* for two terms s and t is represented by $s =? t$.

A *multiset* is an unordered collection with possible duplicate elements. We denote the number of occurrences of an object x in a multiset M by $M(x)$, and define the multiset union $M \cup N$ as the multiset Q such that $Q(x) = M(x) + N(x)$ for every x .

2.2. Unification of terms

In this section and the next, we present a series of algorithms for unification, each of which returns an *mgu* for two unifiable terms. Our approach will be two-sided: on the one hand we will present a series of practical algorithms, from the “naive” to the more sophisticated (and faster), in pseudo-code suitable for implementing in a programming language; and on the other we will present a “rule-based” approach which serves to clarify the essential properties of the process and also to prove the correctness of some of the practical algorithms.

2.2.1. A naive algorithm

The simplest algorithm for unification is perhaps one that is taught in many introductory courses in AI:

Write down two terms and set markers (e.g., two index fingers) at the beginning of the terms. Then:

1. Move the markers together, one symbol at a time, until both move off the end of the term (**success!**), or until they point to two different symbols;
2. If the two symbols are both non-variables, then **fail**; otherwise, one is a variable (call it x) and the other is the first symbol of a subterm (call it t):
 - (a) If x occurs in t , then **fail**;
 - (b) Otherwise, write down " $x \mapsto t$ " as part of the solution, replace x everywhere by t (including in the solution), and return to (1).

This simple algorithm methodically finds disagreements in the two terms to be unified, and attempts to repair them by binding variables to terms: it fails when function symbols clash, or when an attempt is made to unify a variable with a term containing that variable (which is impossible). Already present in this simple algorithm are several interesting issues:

Implementation: What data structures should be used for terms and substitutions? How should application of a substitution be implemented? What order should the operations be performed in?

Correctness: Does the algorithm always terminate? Does it always produce an *mgu* for two unifiable terms, and fail for non-unifiable terms? Do these answers depend on the order of operations?

Complexity: How much space does this take, and how much time?

In the remainder of this section we will consider these issues in detail while developing our sequence of algorithms.

2.2.2. Unification by recursive descent

If we take our naive algorithm and implement it as simply as possible in a programming language, then we would represent terms using either explicit pointer structures (as in C or Pascal) or built-in recursive data types (as in ML and Lisp), and represent substitutions as lists of pairs of terms. Application of a substitution would involve constructing a new term or replacing a variable with a new term. The left-to-right search for disagreements would then be implemented by recursive descent through the terms as shown in Figure 1.

(In an actual implementation, the case "Unify(t, s)" could be moved up before the first "else if" and simply swap s and t if the former is not a variable.) The only detail that might cause some confusion is the exact method for implementing the composition in the last line. This was described in section 2.1; however, in our naive unification algorithm, we omitted the second and third steps from the informal algorithm for composition, and this may be done as well here, due to a simple but important fact about these algorithms: when a binding $x \mapsto t$ is created and applied, x will never appear in another term considered by the algorithm— x has been "eliminated" and occurs only once, in the solution.

This algorithm is essentially the one first described by Robinson [1965], and has been almost universally used in symbolic computation systems.

global σ : substitution; { Initialized to Id }

Unify(s : term; t : term)

begin

if s is a variable **then** { Instantiate variables }

$s := s\sigma$;

if t is a variable **then**

$t := t\sigma$;

if s is a variable **and** $s = t$ **then**

 { Do nothing }

else if $s = f(s_1, \dots, s_n)$ **and** $t = g(t_1, \dots, t_m)$ for $n, m \geq 0$ **then begin**

if $f = g$ **then**

for $i := 1$ to n **do**

 Unify(s_i, t_i);

else Exit with failure { Symbol clash }

end

else if s is not a variable **then**

 Unify(t, s);

else if s occurs in t **then**

 Exit with failure; { Occurs check }

else $\sigma := \sigma\{s \mapsto t\}$;

end;

Figure 1: Unification by recursive descent

2.2.3. A rule-based approach \mathcal{U}

In order to explore some of the logical properties of this algorithm, we now present a simple inference system for deriving solutions for unification problems.

An idempotent substitution $\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ may be represented by a set of equations $\{x_1 \approx t_1, \dots, x_n \approx t_n\}$ in *solved form*, i.e., where each x_i has a single occurrence in the set. For any idempotent substitution σ , the corresponding solved form set will be denoted by $[\sigma]$, and for any set of equations S in solved form, the corresponding substitution will be denoted by σ_S .

A *system* is either the symbol \perp (representing failure) or a pair consisting of a multiset P of unification problems and a set S of equations in solved form. We will use Γ to denote an arbitrary system. A substitution is said to be a unifier (or solution) of a system $P; S$ if it unifies each of the equations in P and S ; the system \perp has no unifiers.

The inference system \mathcal{U} consists of the following transformations on systems:²

²The symbol \cup below when applied to P is *multiset union*.

Trivial:

$$\{s \stackrel{?}{=} s\} \cup P'; S \implies P'; S$$

Decomposition:

$$\{f(s_1, \dots, s_n) \stackrel{?}{=} f(t_1, \dots, t_n)\} \cup P'; S \implies \{s_1 \stackrel{?}{=} t_1, \dots, s_n \stackrel{?}{=} t_n\} \cup P'; S$$

(Note that possibly $n = 0$.)

Symbol Clash:

$$\{f(s_1, \dots, s_n) \stackrel{?}{=} g(t_1, \dots, t_m)\} \cup P'; S \implies \perp$$

if $f \neq g$.

Orient:

$$\{t \stackrel{?}{=} x\} \cup P'; S \implies \{x \stackrel{?}{=} t\} \cup P'; S$$

if t is not a variable.

Occurs Check:

$$\{x \stackrel{?}{=} t\} \cup P'; S \implies \perp$$

if $x \in \text{Vars}(t)$ but $x \neq t$.

Variable Elimination:

$$\{x \stackrel{?}{=} t\} \cup P'; S \implies P'\{x \mapsto t\}; S\{x \mapsto t\} \cup \{x \approx t\}$$

if $x \notin \text{Vars}(t)$.

In order to unify s and t , we create an initial system $\{s \stackrel{?}{=} t\}; \emptyset$ and apply successively rules from \mathcal{U} ; we show below that such a process must terminate, producing a terminal system (i.e., to which no rule applies) in the form of \perp or $\emptyset; S$, where S is a solved form system representing the *mgv* of s and t .

The inference system \mathcal{U} is in essence the same algorithm for unification presented by Herbrand (see Appendix 3 in [Herbrand 1971]); more recently, this formulation of the unification process was introduced by Martelli and Montanari [1982] and has gained wide currency as a formalism for representing unification algorithms (see, for example, [Jouannaud and Kirchner 1991, Snyder 1991]).

Before considering \mathcal{U} *per se*, let us consider how this set of transformations might simulate the actions of the recursive descent algorithm. Suppose we were to print out a trace of the terms s and t , and the global substitution σ , just before the third *if*-statement in *Unify*, e.g.,

$$\begin{array}{lll}
 s_1 & t_1 & Id \\
 s_2 & t_2 & \sigma_2 \\
 s_3 & t_3 & \sigma_3 \\
 \dots & &
 \end{array}$$

This sequence can be simulated by a sequence of transformations

$$\begin{array}{l}
 \{s_1 =^? t_1\}; \emptyset \\
 \Rightarrow \{s_2 =^? t_2\} \cup P_2; S_2 \\
 \Rightarrow \{s_3 =^? t_3\} \cup P_3; S_3 \\
 \Rightarrow \dots
 \end{array}$$

where each $s_i =^? t_i$ is the equation acted on by the rule, and each σ_i is identical to σ_{s_i} . Furthermore, if the call to Unify ends in failure, then the transformation sequence ends in \perp ; and if the call to Unify terminates with success, with a global substitution σ_n , then the transformation sequence ends in a system $\emptyset; S$ where $\sigma_s = \sigma_n$. This simulation can be achieved by treating the multiset P as a stack, always applying a rule to the top equation, and only using Trivial when s is a variable; there is only one possible rule to apply at each step under this control strategy.

Therefore, \mathcal{U} can be viewed as an abstract version of the recursive descent algorithm, and can be used to prove its correctness. In fact, \mathcal{U} has many interesting features in its own right, as we now proceed to show.

2.2.4. Technical results about \mathcal{U}

In this section we present a number of results about \mathcal{U} , adapted from Martelli and Montanari [1982]. Perhaps the simplest property to show is termination.

2.2. LEMMA. *For any finite multiset of equations P , every sequence of transformations in \mathcal{U}*

$$P; \emptyset \Rightarrow P_1; S_1 \Rightarrow P_2; S_2 \Rightarrow \dots$$

terminates either with \perp or with $\emptyset; S$, with S in solved form.

PROOF. Define a complexity measure $\langle n_1, n_2, n_3 \rangle$ on multisets of equations, ordered by the (well-founded) lexicographic ordering on triples of natural numbers, where

n_1 = The number of distinct variables in P ;

n_2 = The number of symbols in P ; and

n_3 = The number of equations in P of the form $t =^? x$, with t not a variable.

Each rule in \mathcal{U} reduces the complexity of the problem P . Furthermore, any equation must fit into one of the cases mentioned on the left-hand sides of the rules, so that a rule can always be applied to a system with non-empty P . Thus, a system to which no rule applies must be in the form \perp or $\emptyset; S$. Since whenever an equation is added to S , the variable on the left-side is eliminated from the rest of the system, each of the systems S_1, S_2, \dots, S must be in solved form. \square

Another interesting fact is that a solution σ produced by \mathcal{U} is always idempotent.

2.3. COROLLARY. *If $P; \emptyset \Rightarrow^+ \emptyset; S$, then σ_s is idempotent.*

One of the most interesting features of \mathcal{U} is that its rules do not change the set of unifiers of a system. The main correctness results about \mathcal{U} are essentially corollaries of this fact.

2.4. LEMMA. *For any transformation $P; S \Rightarrow \Gamma$, a substitution θ unifies $P; S$ iff it unifies Γ .*

PROOF. The only non-trivial cases concern Occurs Check and Variable Elimination. If x occurs in, but is not equal to, t , then clearly x contains fewer symbols than t ; but then $x\theta$ must also contain fewer symbols than $t\theta$, so that x and t can have no unifier.

Regarding Variable Elimination, we know that $x\theta = t\theta$, from which (by structural induction) we can show that

$$u\theta = (u\{x \mapsto t\})\theta$$

for any term u , or indeed for any equation or multiset of equations. But then

$$P'\theta = P'\{x \mapsto t\}\theta \quad \text{and} \quad S\theta = S\{x \mapsto t\}\theta$$

from which the result follows. \square

The first of our major results about \mathcal{U} shows that it does indeed produce a unifier.

2.5. THEOREM. (*Soundness*) *If $P; \emptyset \Rightarrow^+ \emptyset; S$, then σ_s unifies every equation in P .*

PROOF. Note that σ_s unifies S , because it is idempotent; a simple induction with lemma 2.4 shows that σ_s must unify the equations in P . \square

Our second major result shows that \mathcal{U} is able to calculate an *mg*u for two unifiable terms.

2.6. THEOREM. (*Completeness*) *If θ unifies every equation in P , then any maximal sequence of transformations*

$$P; \emptyset \Rightarrow \dots$$

must end in some system $\emptyset; S$ such that $\sigma_s \leq \theta$.

PROOF. Lemmas 2.2 and 2.4 show that such a sequence must end in some terminal system $\emptyset; S$ where θ unifies S . Now for every binding $x \mapsto t$ in σ_s ,

$$x\sigma_s\theta = t\theta = x\theta,$$

and for every $x \notin \text{Dom}(\sigma_s)$, $x\sigma_s\theta = x\theta$, so that $\theta = \sigma_s\theta$. \square

An immediate consequence of these two results is the following.

2.7. COROLLARY. *If P has no unifier, then any maximal transformation sequence from $P; \emptyset$ must have the form*

$$P; \emptyset \Rightarrow \dots \Rightarrow \perp.$$

The most interesting feature of this proof (and the reason for the emphasis on the word “any”) is that the choice of a rule to apply at any stage of the computation is *don't care non-deterministic*, which implies that any control strategy will result in an *mg*u for two unifiable terms, and failure for two non-unifiable terms. Thus, any practical unification algorithm which proceeds by performing the atomic actions of \mathcal{U} , in any order, is sound and complete, and in particular it generates idempotent *mg*us for unifiable terms. However, some sequences of these basic operations may be longer than others, or create larger terms, and not all sequences end in the same exact *mg*u. Before considering the issue of complexity in detail, we digress for a moment to consider this last point.

2.2.5. Some properties of MGU's

Theorem 2.6 shows that any substitution produced by \mathcal{U} (or any algorithm that \mathcal{U} can simulate) is a compact representation of the (infinite) set of all unifiers, which could be generated by composing all possible substitutions with the *mg*u. This means that no information is lost in symbolic computation systems (such as first-order theorem provers and logic-programming interpreters) in solving a unification subproblem and applying the solution to the rest of the computation (this is what happens, in fact, during the unification process itself).

The inference system \mathcal{U} , starting from a single pair of terms s and t , could produce (finitely) many different terminal forms, corresponding to distinct *mg*us for s and t . What is the relationship of these distinct *mg*us? Are there other *mg*us than these? Is there an infinite number? The key to answering these questions lies in the concept of a variable renaming, defined in section 2.1: if σ and θ are both *mg*us of s and t , then $\sigma \doteq \theta$, i.e., they are instances of each other, and hence $\sigma = \theta\rho$ for some variable renaming ρ (for a proof, see [Lassez et al. 1987].)

This means that the set of *mg*us of two terms can be generated from a single *mg*u, by composition with variable renamings (which is a special case of the fact that the set of all unifiers can be generated by composition with arbitrary substitutions). By such an operation, it is possible to create an infinite number of idempotent *mg*us and an infinite number of non-idempotent *mg*us; the finite search tree generated by \mathcal{U} is not able to construct any arbitrary *mg*u, nor even every idempotent *mg*u.

An oft-repeated phrase in the literature states that “*mg*us are unique up to renaming”; the reader should now understand that this vague statement should more properly be: “*mg*us are unique up to composition with a variable renaming.”

This brief exposition of some of the important properties of *mg*us should convince the reader that the collection of all unifiers of two terms has non-trivial properties; later on in this chapter we shall examine the even more complex case of sets of unifiers for E -unification problems. For further characterizations of the set of *mg*us produced by \mathcal{U} , and on unifiers in general, see [Lassez et al. 1987, Eder 1985].

2.2.6. Complexity of recursive descent

This section will begin to consider the complexity of the unification process, a question which will motivate the consideration of further, more sophisticated algorithms for unification.

The approaches to unification so far considered, unfortunately, can take exponential time and space.

2.8. EXAMPLE.

$$h(x_1, x_2, \dots, x_n, f(y_0, y_0), \dots, f(y_{n-1}, y_{n-1}), y_n)$$

and

$$h(f(x_0, x_0), f(x_1, x_1), \dots, f(x_{n-1}, x_{n-1}), y_1, \dots, y_n, x_n)$$

Unifying these two terms will create an *mgu* where each x_i and each y_i is bound to a term with $2^{i+1} - 1$ symbols. Clearly the problem is that the substitution contains many duplicate copies of the same subterms. One idea that might help here would be to represent substitutions as “triangular forms.” Thus,

$$[y_0 \mapsto x_0; y_n \mapsto f(y_{n-1}, y_{n-1}); y_{n-1} \mapsto f(y_{n-2}, y_{n-2}); \dots]$$

would be a triangular form unifier of the two terms. Building up such a substitution during unification consists of simply collecting a list of bindings; no duplicate terms are created, and hence triangular form unifiers can be no larger than the original problem.

Unfortunately, this good idea is not sufficient to rescue the algorithm, as it appears that substitution, and hence the duplication of subterms, is necessary in the terms themselves: in the example, the call to *Unify* on the last arguments, x_n and y_n , which by then are bound to terms with $2^{n+1} - 1$ symbols, will lead to an exponential number of recursive calls. The solution to this problem is to develop a more subtle data structure for terms, and a different method for applying substitutions.

2.3. Unification of term dags

In this section, we consider two approaches to speeding up the unification process. The first approach, which we adapt from Corbin and Bidoit [1983], fixes the problem of duplication of subterms created by substitution by using a graph representation of terms which can share structure; this results in a quadratic algorithm. To develop an asymptotically faster algorithm, however, it is necessary to abandon the recursive descent approach, and recast the problem of unification as the construction of a certain kind of equivalence relation on graphs. This second approach is due to Huet [1976].

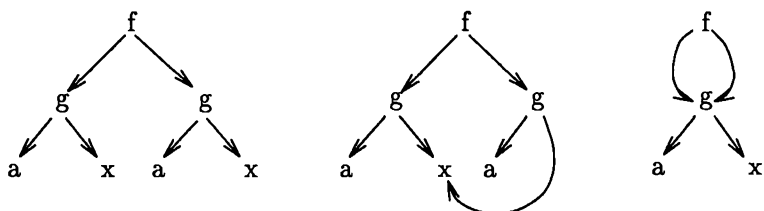
2.3.1. Term dags and substitution

Concerning example 2.8, it should be remarked that the explosion in the size of the terms occurred precisely because there were duplicate occurrences of the same

variables, which cause a duplication of ever larger and larger terms. In order to fix this problem, it is necessary to consider in detail how to represent terms as explicit graphs which share subterms.

2.9. DEFINITION. A *term dag* is a directed, acyclic graph whose nodes are labeled with function symbols, constants, or variables, whose outgoing edges from any node are ordered, and where the outdegree of any node labelled with a symbol f is equal to the arity of f (variables have outdegree 0).

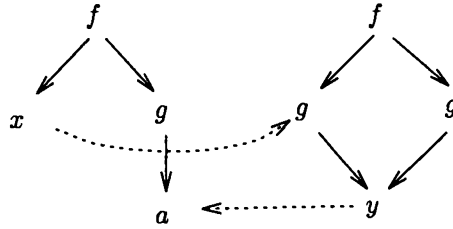
In such a graph, each node has a natural interpretation as a term, and we shall speak of nodes and terms as if they were one and the same (e.g., a “node” $f(a, x)$ is one labeled with f and having arcs to nodes a and x). The only difference between various *dags* representing a particular term is the amount of *structure sharing* among the subterms. For example, we could represent the term $f(g(a, x), g(a, x))$ by any of the following *dags*:



Assuming that names of symbols are strings of characters, it is possible to create a *dag* with unique, shared occurrences of variables in $O(n)$, where n is the number of all characters in the string representation of a unification problem. For example, one can use a *trie* to store the variable names when parsing the terms, so that duplicate occurrences of variables can be pointed to a unique, shared representation of the variable. In the normal case, names have a constant size, and n just represents the number of symbols in the term; we make this assumption in what follows.

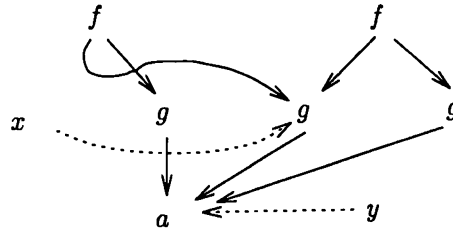
Therefore, we assume that the input to our algorithm is a term *dag* representing the two terms to be unified, with unique, shared occurrences of all variables. We also assume that each node t has an attribute $parents(t)$ which is a list of all parents of t in the graph (i.e., all nodes p which point to t), but do not show these in the diagrams below for simplicity. Parent pointers are necessary when sharing nodes in the dag.

A substitution involving only the subterms of a term *dag* can be represented directly by a relation on the nodes of the *dag*, either stored explicitly as a list of pairs of pointers to nodes, or by storing a link (we will call these *substitution arcs*) in the graph itself, and maintaining a list of variables (nodes) bound by the substitution. Application of such a substitution can be implicit or explicit, the latter involving actual moving of subterm links. For example, two terms $f(x, g(a))$ and $f(g(y), g(y))$, and their *mgu* $\{x \mapsto g(a), y \mapsto a\}$ can be represented by the *dag*:



The *implicit* application of a substitution identifies two nodes connected with a substitution arc, without actually moving any of the subterm links; the binding for a variable can be determined by traversing the graph depth first, left to right. This essentially represents the triangular form (e.g., $[x \mapsto g(y); y \mapsto a]$) in the *dag*. We use this form of substitution in the algorithm of section 2.3.3.

The *explicit* application of a substitution expresses the substitution of binding for variable by moving any arc (subterm or substitution) pointing to a variable to point to the binding. For example,



This represents the “functional” form $\{x \mapsto g(a), y \mapsto a\}$ of the substitution in a direct way. We shall use this explicit form of application in the next algorithm.

2.3.2. Recursive descent on term dags

In this section we present the first algorithm which uses term *dags*. If we think about tracing the operation of the recursive descent algorithm on this new data structure, it might appear that the source of exponential blowup has been removed, since substitution does not duplicate terms. However, it still may be possible to have duplicate *calls* to the same term; in example 2.8, for instance, the terms bound to x_n and y_n (see fig. 2) will be unified when x_0 is bound to y_0 ; however, the recursive descent algorithm will then blithely explore every other path through the pair of terms, resulting in an exponential number of recursive calls.

Clearly, we need to keep from revisiting already-solved problems in the graph. The best solution is simply to do structure sharing “on the fly” by merging unified terms (which are, after all, now identical), and then checking for identity of nodes in the first step. Merging two nodes s and t in a graph Δ can be implemented by moving arcs. Let $\text{parents}(s) = \{p_1, \dots, p_n\}$; then

1. For each p_i , replace the subterm arc $p_i \rightarrow s$ by $p_i \rightarrow t$;
2. Let $\text{parents}(t) := \text{parents}(s) \cup \text{parents}(t)$; and
3. Let $\text{parents}(s) := \emptyset$.

This shares the structure of t and isolates the node s . In the algorithm below, we will denote by $\text{Replace}(\Delta, s, t)$ the new graph created from a graph Δ by merging

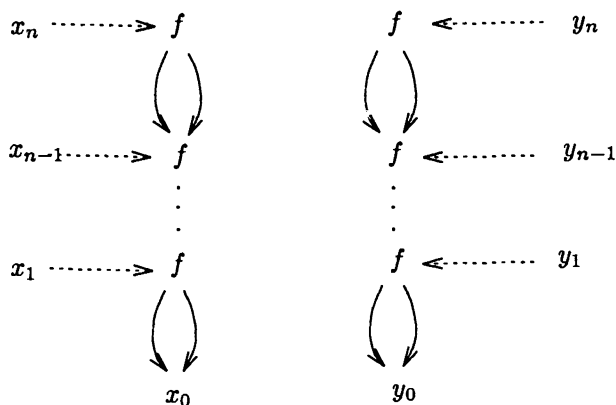


Figure 2: A dag representation of the terms bound to x_n and y_n in example 2.8.

s and t in this fashion.

The algorithm takes as input a term *dag* in which all occurrences of variables are shared (i.e., each variable occurs exactly once). Even with these additions, our recursive descent algorithm is mostly unchanged:

global Δ : termDag; { Term *dag* for s and t with shared variables }
 global σ : list of pairs of nodes; { Initialized to empty }

```

UnifyDag(  $s$  : node;  $t$  : node )
  begin
    if  $s$  and  $t$  are the same node then
      { Do nothing }
    else if  $s = f(s_1, \dots, s_n)$  and  $t = g(t_1, \dots, t_m)$  then begin
      if  $f = g$  then
        for  $i := 1$  to  $n$  do
          UnifyDag(  $s_i, t_i$  );
        else Exit with failure { Symbol clash }
      end
    else if  $s$  is not a variable then
      Unify(  $t, s$  );
    else if  $s$  occurs in  $t$  then
      Exit with failure; { Occurs check }
    else
      Add ( $s, t$ ) to the end of the list  $\sigma$ ;
       $\Delta := \text{Replace}(\Delta, s, t)$ ; { Since they are now unified }
    end;
  end;

```

The occurs check is implemented as a standard graph traversal to search for the given node s below t by following subterm arcs.

The correctness of the data structure for this algorithm is dependent on the following result from Corbin and Bidoit [1983], which can be proved by induction on the *dag*.

2.10. LEMMA. *Let Δ be a term dag with nodes x and t such that there is no path from t to x .*

- *Replace(Δ, x, t) is an acyclic graph containing the same nodes (with the same labels) as Δ .*
- *Consider a distinguished node in Δ corresponding to the term s , and let s' be the term corresponding to the same node in Replace(Δ, x, t); then:*
 - *if $s = x$, then $s' = x$;*
 - *otherwise, $s' = s\{x \mapsto t\}$.*

In order to prove soundness and completeness, we may again show that \mathcal{U} can “trace” the terms in each call to UnifyDags, the only difference being that when Trivial is used, s may not necessarily be a variable (i.e., when UnifyDag is called on two terms previously unified, and hence shared as one node). From a logical point of view (thinking in term of the symbolic expressions being manipulated), nothing has changed—only the underlying data structure for terms and substitutions.

Thus, the only thing that remains to be considered is the complexity of UnifyDag. Since each call to this function isolates a node, there can not be more than n calls *in toto* (where n is the number of symbols occurring in the original terms). Each call does a constant amount of work except for the occurs check (which traverses no more than n nodes) and the moving of no more than n pointers. Maintaining the lists of parents costs $O(n)$ at each call. The original construction of the *dag* takes $O(n)$. This results in a time complexity of $O(n^2)$; clearly the space used is $O(n)$.

2.3.3. An almost-linear algorithm

It would be possible to speed up this algorithm by making changes to the way substitutions are represented (see [Baader and Siekmann 1994]), however, we will now consider an alternate approach which gives more insight into the nature of unification. This approach makes the following fundamental changes to the approach considered so far:

- instead of recursive calls to pairs of subterms which must be unified, we will recast the problem as that of constructing an equivalence relation whose classes are terms that must be unified;
- substitution will (in some sense) be replaced by the union of equivalence classes; and
- the repeated calls to the occurs check will be replaced by a single pass through the graph to check for acyclicity.

The term *dag* data structure will be used for these algorithms as well, however, we will not move pointers as in the last section. Instead, we consider the unification problem as one involving the following relation on terms.

2.11. DEFINITION. A *term relation* is an equivalence relation on terms, and is *homogeneous* if no equivalence class contains $f(\dots)$ and $g(\dots)$ with $f \neq g$; it is *acyclic* if no term is equivalent to a proper subterm of itself.

A *unification relation* is a homogeneous, acyclic term relation satisfying the *unification axiom*: For any f and terms s_i and t_i ,

$$f(s_1, \dots, s_n) \cong f(t_1, \dots, t_n) \longrightarrow s_1 \cong t_1 \wedge \dots \wedge s_n \cong t_n.$$

The *unification closure* of s and t , when it exists, is the least unification relation which makes s and t equivalent.

The algorithm presented in this section takes its starting point from the following fact.

2.12. LEMMA. *If s and t are unifiable, then there exists a unification closure for s and t .*

PROOF. For any unifier θ of s and t , define the relation

$$u \cong_{\theta} v \text{ iff } u\theta = v\theta.$$

Clearly this is a unification relation. Since the intersection of two unification relations relating s and t is again a unification relation relating s and t , whenever s and t are unifiable there is a *least* such relation \cong which joins classes only when forced to apply the unification axiom to subterms of s and t . \square

The unification-closure approach to unification, first presented in [Huet 1976], attempts to construct this relation on two terms, which, as we shall show, corresponds to finding an *mgv*. However, before presenting the algorithm, we need a number of ancillary notions.

2.13. DEFINITION. For any term relation \cong , let a *schema function* be a function ς from equivalence classes to terms such that for any class C ,

1. $\varsigma(C) \in C$; and
2. $\varsigma(C)$ is a variable only if C consists entirely of variables.

The term $\varsigma(C)$ will be called the *schema term* for C .

The point here is that the schema term is a functional form whenever such exists, and will be used to propagate information downward using the unification axiom; it is also used to define substitutions. Note that schema functions are not unique, but there always exists at least one for any term relation; we assume in the following that such a function has been chosen for any given unification closure.

Note that for any acyclic term relation there exists a partial ordering \succ such that for any term $f(\dots s \dots)$, we have $[f(\dots s \dots)] \succ [s]$.

2.14. DEFINITION. For any unification closure \cong , define σ_{\cong} by:

$$x\sigma_{\cong} = \begin{cases} y & \text{if } \varsigma([x]) = y \\ f(s_1\sigma_{\cong}, \dots, s_n\sigma_{\cong}) & \text{if } \varsigma([x]) = f(s_1, \dots, s_n) \end{cases}$$

(This notion is well-defined by recursion on the partial order \succ ; $\text{Dom}(\sigma_{\cong})$ is finite because \cong has only a finite number of non-trivial equivalence classes.)

2.15. THEOREM. *Terms s and t are unifiable iff there is a unification closure for s and t . In the affirmative case, σ_{\cong} is an mgu for s and t .*

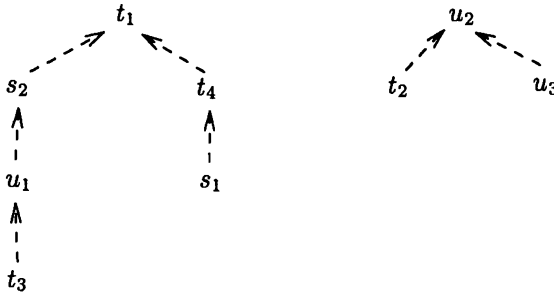
PROOF. The *only if* direction has been proved in our previous lemma. For the other direction, let \cong be a unification closure for s and t . We claim that for every term u , $u\sigma_{\cong} = \varsigma([u])\sigma_{\cong}$ (thus, σ_{\cong} unifies each pair of equivalent terms, in particular s and t), and proceed by induction on the size of u . For the base case, if u is a constant or variable, then the result is trivial by the definition of σ_{\cong} . Now suppose that $u = f(s_1, \dots, s_n)$ and $\varsigma([u]) = f(t_1, \dots, t_n)$; since \cong is closed under the unification axiom, then for each i , $s_i \cong t_i$, and thus by the induction hypothesis, $s_i\sigma_{\cong} = t_i\sigma_{\cong}$.

To prove that σ_{\cong} is an mgu in the affirmative case, we show that for any unifier θ , we have $u\sigma_{\cong}\theta = u\theta$ for any term u , and proceed by induction on \succ . Assume that \cong_{θ} is as defined in the previous lemma. (In the following, ς refers to some fixed schema function for σ_{\cong} .) First, note that if $u \cong v$, then $u\theta = v\theta$, since \cong is contained in \cong_{θ} . Now, for the base case, if $[u]$ contains only constants and variables, then $u\sigma_{\cong} = \varsigma([u]) \cong u$, from which it follows that $u\sigma_{\cong}\theta = u\theta$. For the induction step, it must be the case that $\varsigma([u])$ equals some $f(s_1, \dots, s_n)$, and u is either a term of the form $f(t_1, \dots, t_n)$, or is a variable x . In the first case, $u\sigma_{\cong}\theta = u\theta$ by a direct use of the induction hypothesis. In the second case, $x\sigma_{\cong} = f(s_1\sigma_{\cong}, \dots, s_n\sigma_{\cong})$, and $x\theta = f(s_1, \dots, s_n)\theta$ (since \cong is contained in \cong_{θ}), so that

$$x\theta = f(s_1\theta, \dots, s_n\theta) = f(s_1\sigma_{\cong}\theta, \dots, s_n\sigma_{\cong}\theta) = f(s_1\sigma_{\cong}, \dots, s_n\sigma_{\cong})\theta = x\sigma_{\cong}\theta,$$

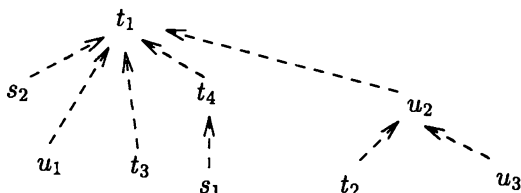
the second step involving the induction hypothesis. \square

This result motivates the design of an efficient unification algorithm which attempts to build a unification closure for two terms, and then extracts the mgu. To do this, it is necessary to have some means for maintaining equivalence classes and for applying the unification axiom to classes; the most efficient data structure represents classes as trees of *class pointers* (which we represent by dashed lines) with a class *representative* at the root:



To determine whether two terms are equivalent, it is only necessary to find the roots of the trees and check for identity; and to join two classes, one class is made

a subtree of the other's root. To reduce the height of the trees as much as possible, two subtle refinements are made: (i) maintain a count of the size of each class in the representative, and when joining classes, make the smaller one a subtree of the larger; and (ii) when following paths to the root to determine equivalence, compress the paths by pointing all nodes encountered directly at the root. For example, if we wished to take the union of the two classes $[t_3]$ and $[u_3]$, we would find the representatives for the two classes, compressing the path from t_3 , and then add a class link from the representative of the smaller class to the larger:



Such a data structure can process a series of $O(n)$ Unions and Finds in $O(n\alpha(n))$, where α is the functional inverse of Ackermann's function, and which, for all *practical* purposes, may be considered as a small constant factor.

The term *dag* for this approach needs no parent pointers, as in the previous algorithm, but does need

- *class* pointers;
- a counter of the *size* of the class stored in the representative;
- a pointer from each representative to the *schema* term for the class;
- boolean flags *visited* and *acyclic* in each node used in cycle checking (both initialized to **false**);
- a pointer *vars* from each representative to a list of all variables in the class (used when generating solutions).

Note that maintaining lists of parents of each node is not necessary in this algorithm. A representative is simply a node whose class pointer points to itself. The algorithm based on this approach may now be given. It is shown in Figures 3 and 4. The term *dag* Δ for s and t is initialized to the identity relation, where each class contains a single term; thus for each node the *class* and *schema* pointers are initialized to point to the same node, and the *size* is initialized to 1. The *vars* list is initialized to empty for non-variable nodes, and to a singleton list for variable nodes.

If $\text{Unify}(s, t)$ does not fail, then σ contains a triangular form solution. Find-Solution attempts to find such a solution, and fails iff there exists a cycle in the graph. (We are essentially traversing the common term $s\sigma$ by replacing s by its schema term in the first line.) The fields *visited* and *acyclic* are both necessary, the first to find a cycle in the current exploration path, and the second to keep from reexamining nodes which have already been excluded from any possible cycles.

The correctness of this method depends on verifying that it implements correctly the construction of an acyclic unification closure. The essential points are that

- the equivalence is clearly homogeneous;
- equivalence classes are joined iff required by the unification axiom, hence the relation is *least*;

global Δ : termDag; { Term *dag* for *s* and *t* with shared variables }
 global σ : list of bindings := nil; { Triangular form solution }

Unify(*s* : node; *t* : node)

begin

 UnifClosure(*s*, *t*);

 FindSolution(*s*);

end;

UnifClosure(*s* : node; *t* : node)

begin

s := Find(*s*); { Find representatives }

t := Find(*t*);

if *s* and *t* are the same node **then**

 { Do nothing }

else begin

if $\varsigma([s]) = f(s_1, \dots, s_n)$ and $\varsigma([t]) = g(t_1, \dots, t_m)$ for $n, m \geq 0$

then begin

if $f = g$ **then begin**

 Union(*s*, *t*);

for $i := 1$ to n **do**

 UnifClosure(*s_i*, *t_i*);

end

else Exit with failure { Symbol clash }

end

else Union(*s*, *t*);

end;

end;

Union(*s* : node; *t* : node) { *s* and *t* are representatives }

begin

if $\text{size}(s) \geq \text{size}(t)$ **then begin**

$\text{size}(s) := \text{size}(s) + \text{size}(t)$;

$\text{vars}(s) := \text{concatenation of lists vars}(s) \text{ and vars}(t)$;

if $\varsigma([s])$ is a variable **then**

$\varsigma([s]) := \varsigma([t])$;

$\text{class}(t) := s$;

end

else begin

$\text{size}(t) := \text{size}(t) + \text{size}(s)$;

$\text{vars}(t) := \text{concatenation of lists vars}(t) \text{ and vars}(s)$;

if $\varsigma([t])$ is a variable **then**

$\varsigma([t]) := \varsigma([s])$;

$\text{class}(s) := t$;

end;

end;

Figure 3: Unification algorithm

```

Find(  $s$  : node ) { Returns representative for  $[s]$  and compresses paths }
   $t$  : node;
  begin
    if  $\text{class}(s) = s$  {  $s$  is a representative } then
      Return  $s$ ;
    else begin
       $t := \text{Find}(\text{class}(s))$ ;
       $\text{class}(s) := t$ ;
      return  $t$ ;
    end;
  end;

FindSolution( $s$  : node); { Fails if exists a cycle below  $s$  }
  begin;
     $s := \varsigma(\text{Find}(s))$ ;
    if  $\text{acyclic}(s)$  then
      Return; {  $s$  is not part of a cycle }
    if  $\text{visited}(s)$  then
      Fail; { Exists a cycle }
    if  $s = f(s_1, \dots, s_n)$  for some  $n > 0$  then begin
       $\text{visited}(s) := \text{true}$ ;
      for  $i := 1$  to  $n$  do
        FindSolution( $s_i$ );
       $\text{visited}(s) := \text{false}$ ;
    end;
     $\text{acyclic}(s) := \text{true}$ ;
    foreach  $x \in \text{vars}(\text{Find}(s))$  do
      if  $x \neq s$  then
        Add  $[x \mapsto s]$  to front of  $\sigma$ ;
  end;

```

Figure 4: Unification algorithm, continued

- FindSolution fails iff there is a cycle in the graph; and
- whenever a binding $[x \mapsto s]$ is added to σ , all relevant bindings for variables in s already occur in σ .

The complexity of the algorithm is $O(n\alpha(n))$, as, with the exception of **Find**, each function can be called at most n times for terms with n symbols, and each call performs a constant amount of work (note that the work of concatenating the *vars* lists can be accomplished in $O(n)$ if pointers to the last cell in the list are kept, and concatenation is performed by moving pointers rather than by the standard append operation). The dominating cost is therefore the calls to **Find**, which, as mentioned above, can cost $O(n\alpha(n))$.

Linear-time algorithms for unification have been presented by Paterson and Wegman [1978] (cf. [Champeaux 1986]) and Martelli and Montanari [1982], to which

we refer the reader for further study.

3. Equational unification

Like syntactic unification, equational unification is concerned with the problem of making terms equal by applying a suitable substitution. The only difference is that syntactic equality is replaced by equality modulo an equational theory E . At first sight, one might think that this is minor change, and that the notions and approaches from syntactic unification can easily be adapted to this new situation. It turns out, however, that equational unification requires some non-trivial adjustments of the basic notation. In particular, the notion of a most general unifier is no longer sufficient for the purpose of representing all unifiers since there may exist E -unifiable terms that do not have a most general E -unifier. In the first subsection, we introduce the basic notions as they are currently used in unification theory, and in the subsequent subsection, we point out some differences to the case of syntactic unification, and explain the reason for introducing the notions in this modified way. The third subsection introduces order-theoretic, logical, algebraic, and category-theoretic reformulations of some of these notions. We conclude the section with a short survey of results in unification theory. Some of these results will be treated in more detail in subsequent sections.

3.1. Basic notions

An *equational theory* is defined by a *set of identities* E , i.e., a subset of $\mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{V})$ for a signature \mathcal{F} and a (countably infinite) set of variables \mathcal{V} . It is the least congruence relation on the term algebra $\mathcal{T}(\mathcal{F}, \mathcal{V})$ that is closed under substitution and contains E , and it will be denoted by $=_E$ (see [Dershowitz and Plaisted 2001, page 575] (Chapter 9 of this Handbook) for a more detailed definition of the relation $=_E$). Identities are written in the form $s \approx t$. If $s =_E t$, then we say that the term s is *equal modulo E* to the term t . For example, let f be a binary function symbol. The identity $C := \{f(x, y) \approx f(y, x)\}$ says that f is commutative, and the identity $A := \{f(f(x, y), z) \approx f(x, f(y, z))\}$ expresses associativity of f . We have $f(f(a, b), c) =_C f(c, f(b, a))$, and $f(a, f(x, b)) =_A f(f(a, x), b)$. In the following, we will often slightly abuse the notion of an equational theory by also calling a set of defining identities E an equational theory. For a given set of identities E , we denote by $\text{Sig}(E)$ the set of all function symbols occurring in E .

3.1. DEFINITION. Let E be an equational theory and \mathcal{F} a signature containing $\text{Sig}(E)$. An *E -unification problem* over \mathcal{F} is a finite set of equations

$$\Gamma = \{s_1 \stackrel{?}{=}_E t_1, \dots, s_n \stackrel{?}{=}_E t_n\}$$

between \mathcal{F} -terms with variables in a (countably infinite) set of variables \mathcal{V} . An *E -unifier* of Γ is a substitution σ such that $s_1\sigma =_E t_1\sigma, \dots, s_n\sigma =_E t_n\sigma$. The set of

all E -unifiers of Γ is denoted by $\mathcal{U}_E(\Gamma)$, and Γ is E -unifiable iff $\mathcal{U}_E(\Gamma) \neq \emptyset$.

Obviously, syntactic unification is the special case of this definition where $E = \emptyset$. Any syntactic unifier of an E -unification problem Γ is also an E -unifier, but for $E \neq \emptyset$, the set $\mathcal{U}_E(\Gamma)$ may have additional elements. For example, if a and b are distinct constant symbols, then the C -unification problem $\{f(a, x) =_C^? f(b, y)\}$ has $\{x \mapsto b, y \mapsto a\}$ as C -unifier, whereas the terms $f(a, x)$ and $f(b, y)$ do not have a syntactic unifier. For the A -unification problem $\Gamma := \{f(a, x) =_A^? f(y, b)\}$, the set $\mathcal{U}_A(\Gamma)$ contains the syntactic unifier $\{x \mapsto b, y \mapsto a\}$ of $f(a, x)$ and $f(y, b)$, but also additional A -unifiers such as $\{x \mapsto f(z, b), y \mapsto f(a, z)\}$.

The instantiation quasi-ordering \leq on substitutions is adapted to the case of equational unification as follows:

3.2. DEFINITION. Let E be an equational theory and \mathcal{X} a set of variables. The substitution σ is *more general modulo E on \mathcal{X}* than the substitution θ iff there exists a substitution λ such that $x\theta =_E x\sigma\lambda$ for all $x \in \mathcal{X}$. In this case we write $\sigma \leq_E^\mathcal{X} \theta$ and say that θ is an E -instance of σ on \mathcal{X} .

It is easy to see that $\leq_E^\mathcal{X}$ is a quasi-ordering, i.e., a reflexive and transitive binary relation. The associated equivalence is denoted by $\dot{=}^\mathcal{X}_E$, i.e., $\sigma \dot{=}^\mathcal{X}_E \theta$ iff $\sigma \leq_E^\mathcal{X} \theta$ and $\theta \leq_E^\mathcal{X} \sigma$.

When comparing E -unifiers of a problem Γ , the set \mathcal{X} is the set of all variables occurring in Γ . Unlike the case of syntactic unification, unifiable E -unification problems need not have a most general E -unifier. For example, the C -unification problem $\{f(x, y) =_C^? f(a, b)\}$ has the two C -unifiers $\sigma_1 := \{x \mapsto a, y \mapsto b\}$ and $\sigma_2 := \{x \mapsto b, y \mapsto a\}$. On $\text{Var}(\Gamma) = \{x, y\}$, any C -unifier of Γ is equal to either σ_1 or σ_2 , and σ_1 and σ_2 are not comparable w.r.t the instantiation quasi-ordering $\leq_C^{\{x, y\}}$. Consequently, there cannot be a most general C -unifier of Γ . Thus, the rôle of the most general unifier must in general be taken on by a complete set of unifiers.

3.3. DEFINITION. Let Γ be an E -unification problem over \mathcal{F} and let $\mathcal{X} := \text{Var}(\Gamma)$ be the set of all variables occurring in Γ . A *complete set of E -unifiers* of Γ is a set \mathcal{C} of substitutions such that

1. $\mathcal{C} \subseteq \mathcal{U}_E(\Gamma)$, i.e., each element of \mathcal{C} is an E -unifier of Γ ,
2. for each $\theta \in \mathcal{U}_E(\Gamma)$ there exists $\sigma \in \mathcal{C}$ such that $\sigma \leq_E^\mathcal{X} \theta$.

The set \mathcal{C} is a *minimal complete set of E -unifiers* of Γ iff it is a complete set that satisfies

3. two distinct elements of \mathcal{C} are incomparable w.r.t. $\leq_E^\mathcal{X}$, i.e., for all $\sigma, \sigma' \in \mathcal{C}$, $\sigma \leq_E^\mathcal{X} \sigma'$ implies $\sigma = \sigma'$.

The substitution σ is a *most general E -unifier* of Γ iff $\{\sigma\}$ is a (minimal) complete set of E -unifiers of Γ .

If the unification problem Γ is not E -unifiable, then the empty set is a minimal complete set of E -unifiers of Γ . Depending on the equational theory E , minimal complete sets of E -unifiers need not always exist, and even if they do, they may be

infinite (see below). It is, however, easy to show that they are unique up to instantiation equivalence $\stackrel{\cdot \chi}{=}_E$ (see subsection 3.3.1). This makes sure that the following definition of the unification type of an E -unification problem and of an equational theory E is unambiguous.

3.4. DEFINITION. Let E be an equational theory, and let Γ be an E -unification problem over \mathcal{F} . The problem Γ has type *unitary* (*finitary*, *infinitary*) iff it has a minimal complete set of E -unifiers of cardinality 1 (finite cardinality, infinite cardinality). If Γ does not have a minimal complete set of E -unifiers, then it is of type *zero*. We abbreviate type unitary by 1, type finitary by ω , type infinitary by ∞ , and type zero by 0, and order these types as follows: $1 < \omega < \infty < 0$.

The *unification type* of E w.r.t. the signature \mathcal{F} is the maximal type of an E -unification problem over \mathcal{F} .

According to this definition, an equational theory that is unitary is *not* finitary, and a theory of type zero is *not* infinitary. In the literature, these notions have sometimes been defined such that unitary implies finitary (i.e., unitary theories are a special case of finitary theories) and type zero implies infinitary. We prefer a stricter separation between the types. In order to express that a theory is unitary or finitary (in the sense of definition 3.4) we say that it is *at most finitary*. Analogously, to express that a theory is infinitary or of type zero we say that it is *at least infinitary*.

It should also be noted that the unification type of an equational theory depends not only on E , but also on the set of function symbols \mathcal{F} that are allowed to occur in the unification problems (see subsection 3.2.2 for more details). We provide an example for each of the four types.

3.5. EXAMPLE (*unitary*). Since any unifiable unification problem has a most general syntactic unifier, the empty theory \emptyset (which obviously defines syntactic equality) has unification type unitary w.r.t. any signature \mathcal{F} .

3.6. EXAMPLE (*finitary*). Above, we have seen that commutativity C is not unitary since the C -unification problem $\{f(x, y) \stackrel{?}{=}_C f(a, b)\}$ does not have a most general C -unifier. It is not hard to show that C is finitary w.r.t. any signature \mathcal{F} . In fact, the C -equivalence class $[t]_C := \{t' \mid t \stackrel{?}{=}_C t'\}$ of a given term t is easily shown to be finite. For a given C -unification problem $\Gamma = \{s_1 \stackrel{?}{=}_C t_1, \dots, s_n \stackrel{?}{=}_C t_n\}$, we consider all possible syntactic unification problems of the form $\Gamma' = \{s'_1 \stackrel{?}{=} t'_1, \dots, s'_n \stackrel{?}{=} t'_n\}$ where $s_i \stackrel{?}{=}_C s'_i$ and $t_i \stackrel{?}{=}_C t'_i$ for all $i, 1 \leq i \leq n$. There are only finitely many such sets Γ' , and it can be shown that the collection of all the syntactic most general unifiers of these sets is a complete set of C -unifiers of Γ [Siekman 1979]. In most cases, this set is not minimal, but obviously a minimal complete set can be obtained by eliminating redundant elements, i.e., elements that are C -instances of other elements of the set.

3.7. EXAMPLE (*infinitary*). Even though associativity A is similar to C in that A -equivalence classes are finite, the unification method outlined for C does not work

for A . It is easy to see that the A -unification problem $\{f(a, x) =_A^? f(x, a)\}$ has an infinite minimal complete set of A -unifiers, namely $\{\sigma_n \mid n \geq 1\}$, where for each n the substitution $\sigma_n := \{x \mapsto f(a, f(a, \dots, f(a, a) \dots))\}$ replaces x by a term containing n occurrences of the constant a . Consequently, A cannot be unitary or finitary. Plotkin [1972] describes a procedure that generates a minimal complete set of A -unifiers of a given A -unification problem over an arbitrary set of function symbols \mathcal{F} , which shows that A is in fact infinitary and not of type zero.

3.8. EXAMPLE (zero). The first example of an equational theory of unification type zero was described by Fages and Huet [1983] and [1986]. In [Baader 1986] it is shown that the theory of idempotent semigroups, i.e., $AI := A \cup \{f(x, x) \approx x\}$ is of unification type zero since the AI -unification problem $\{f(x, f(y, x)) =_{AI}^? f(x, f(z, x))\}$ does not have a minimal complete set of AI -unifiers. This result was also shown by Schmidt-Schauß [1986b], but his example problem $\{f(z, f(a, f(x, f(a, z)))) =_{AI}^? f(z, f(a, z))\}$ contains an additional constant a .

For syntactic unification, a “unification algorithm” is an algorithm that computes a most general (syntactic) unifier of a given unification problem if it exists, and determines non-unifiability otherwise. For equational unification, this notion must be adapted. More precisely, we are interested in different types of algorithms, depending on what the equational theory allows and what is needed in applications.

An *E-unification algorithm* (w.r.t. \mathcal{F}) is an algorithm that computes a *finite* complete set of E -unifiers for all E -unification problems over \mathcal{F} . Ideally, the computed sets should also be minimal. There are, however, theories for which it is easier to compute a not necessarily minimal set (commutativity C is an example). We call an E -unification algorithm *minimal* iff it computes a finite minimal complete set of E -unifiers. As mentioned in example 3.6, an E -unification algorithm can always be turned into a minimal one by eliminating redundant unifiers, provided that the E -instantiation quasi-ordering is decidable.

In applications such as constraint-based approaches to automated deduction and rewriting (see [Bürckert 1991, Nieuwenhuis and Rubio 1994, Kirchner and Kirchner 1989] and [Nieuwenhuis and Rubio 2001], Chapter 7 of this Handbook), it is not necessary to compute complete sets of unifiers. Instead, it is sufficient to test unification problems for unifiability. An algorithm that is able to decide unifiability of E -unification problems (over \mathcal{F}) is called a *decision procedure* for E -unification (w.r.t. \mathcal{F}). Obviously, any E -unification algorithm yields a decision procedure for E -unification since a given E -unification problem Γ is unifiable iff the computed finite complete set is nonempty.

For theories that are not unitary or finitary, the notion of an E -unification algorithm, as introduced above, is not appropriate. A (*minimal*) *E-unification procedure* is a procedure that enumerates a possibly infinite (minimal) complete set of E -unifiers. The procedure by Plotkin [1972] mentioned in example 3.7 is a minimal A -unification procedure. An E -unification procedure need not yield a decision procedure for E -unification since it need not terminate even if the input problem does not have E -unifiers. This is, e.g., the case for Plotkin's procedure. A -

unification (more precisely, the question whether there exists an A -unifier for a given A -unification problem) is nevertheless decidable, but this is a lot harder to show [Makanin 1977] than designing a minimal A -unification procedure.

3.2. New issues

The notions introduced above deviate in several respects from the notions introduced for syntactic unification. In this subsection, we point out the reasons why this was necessary.

3.2.1. The instantiation quasi-ordering

For syntactic unification, the instantiation quasi-ordering \leq was defined by $\sigma \leq \theta$ iff there exists λ such that $\theta = \sigma\lambda$. In the definition of the instantiation quasi-ordering for E -unification, syntactic equality is (quite naturally) replaced by equality modulo E . What may seem less clear is why we have restricted this equality (modulo E) to the variables occurring in the unification problem. Obviously, the ordering obtained this way is stronger than the one that requires equality on all variables (i.e., more substitutions are comparable). In applications in automated deduction, where substitutions generally have meaning only in the context of the expressions (i.e., unification problems) that produced them, it is admissible to use an ordering that compares alternate solutions only with respect to this small set of variables. It is also advisable, as this stronger ordering allows for smaller minimal complete sets. For example, the theory $ACU := AC \cup \{f(x, e) = x\}$ is known to be unitary w.r.t. $\mathcal{F} := \{f, e\}$. If the weaker instantiation quasi-ordering (i.e., the one comparing substitutions on all variables) were used, this would no longer be true [Baader 1991].

Another difference between the equational case and the syntactic case concerns the characterization of the instantiation equivalence \doteq . For $E = \emptyset$, two substitutions are instantiation equivalent iff they are equal up to composition with a variable renaming. It should be noted that this need no longer be the case for $E \neq \emptyset$, even if one replaces “equal up to composition with a variable renaming” by “equal modulo E up to composition with a variable renaming.” For example, consider the equational theory $I := \{f(x, x) \approx x\}$, and the substitutions $\sigma := \{x \mapsto y\}$ and $\theta := \{x \mapsto f(y, z)\}$. Using the substitutions $\lambda_1 := \{y \mapsto f(y, z)\}$ and $\lambda_2 := \{y \mapsto y, z \mapsto y\}$, it is easy to show that $\sigma \doteq_E^{\{x\}} \theta$. However, a variable renaming cannot identify y and z , and thus $f(y, z)\rho \neq_I y$ for every such renaming ρ .

3.2.2. The signature matters

In the definitions of E -unification problems, unification type, etc., we have always explicitly stated which function symbols may occur in the unification problems. The reason is that the unification properties of an equational theory (like decidability, unification type, etc.) may depend on this set of function symbols. In most cases, however, a less fine-grained distinction is sufficient. Recall that $Sig(E)$ denotes the

set of all function symbols occurring in the equational theory E .

3.9. DEFINITION. Let E be an equational theory and Γ an E -unification problem over \mathcal{F} .

- Γ is an *elementary* E -unification problem iff $\mathcal{F} = \text{Sig}(E)$.
- Γ is an E -unification problem *with constants* iff $\mathcal{F} \setminus \text{Sig}(E)$ is a set of constant symbols.
- In a *general* E -unification problem, $\mathcal{F} \setminus \text{Sig}(E)$ may contain arbitrary function symbols.

Following this distinction, we can introduce three different unification types for an equational theory. The *unification type of E w.r.t. elementary unification* is the maximal unification type of E w.r.t. all sets of function symbols \mathcal{F} satisfying $\mathcal{F} = \text{Sig}(E)$. Accordingly, the *unification type of E w.r.t. unification with constants* is the maximal unification type of E w.r.t. all sets of function symbols \mathcal{F} such that $\mathcal{F} \setminus \text{Sig}(E)$ is a set of constant symbols, and the *unification type of E w.r.t. general unification*³ is the maximal unification type of E w.r.t. all signatures \mathcal{F} . Obviously, the same distinction can be made for decidability of E -unification, and for other interesting properties of E -unification problems. Constant (function) symbols that do not occur in E are called *free* constant (function) symbols w.r.t. E .

The theory ACU introduced above is an example of a theory that is unitary for elementary unification, but only finitary for unification with constants (see, e.g., [Herold and Siekmann 1987]). Bürckert [1989] has shown that there exists an equational theory for which elementary unification is decidable, but unification with constants is undecidable.

Applications of equational unification in automated deduction usually yield general unification problems. For example, in resolution-based theorem proving, the additional free function symbols are often generated by Skolemization.

From a strictly formal point of view, the definition of an E -unifier (see definition 3.1) is ambiguous since it does not specify over which signature the terms that are substituted for the variables may be built. By default, we have assumed that this set is the set \mathcal{F} , which contains all function symbols occurring in E or Γ . One might ask whether there would be a significant difference if we allowed the substitutions to introduce additional free function symbols. It is easy to show, however, that there is no such difference since any E -unifier of Γ that introduces additional free function symbols is an instance of an E -unifier that uses only symbols from \mathcal{F} : this more general unifier can, in principle, be obtained by replacing subterms starting with such additional function symbols by new variables, while taking care that $=_E$ -equal subterms are replaced by the same variable. Thus, if we restrict the set of E -unifiers to substitutions over \mathcal{F} , we obtain a complete set of E -unifiers even w.r.t. substitutions over larger signatures. This justifies the (formally somewhat sloppy) definition of the set of E -unifiers given above.

³It should be noted that this use of the term “general unification” is distinct from the one in [Gallier and Snyder 1989, Snyder 1991], where it refers to methods that provide unification procedures for arbitrary equational theories (see section 4.1).

3.2.3. Single equations versus systems of equations

For syntactic unification, solving a system of term equations can be reduced to solving a single equation $s = ? t$. For this reason, syntactic unification is sometimes only considered for single equations. For equational unification, the same holds if one considers general unification. In fact, if $f \in \mathcal{F}$ is an n -ary function symbol not contained in $\text{Sig}(E)$, then the E -unification problem $\{s_1 = ?_E t_1, \dots, s_n = ?_E t_n\}$ over \mathcal{F} has the same set of unifiers as $\{f(s_1, \dots, s_n) = ?_E f(t_1, \dots, t_n)\}$.

For elementary unification and for unification with constants, however, there may be significant differences. For example, there exists an equational theory E such that all elementary E -unification problems of cardinality 1 (i.e., single equations) have minimal complete sets of E -unifiers, but E is of type zero w.r.t. elementary unification since there exists an elementary E -unification problem of cardinality 2 that does not have a minimal complete set of E -unifiers [Bürckert, Herold and Schmidt-Schauß 1989]. Narendran and Otto [1990] give an example of a theory such that unifiability of elementary unification problems of cardinality 1 is decidable, but unifiability is undecidable for elementary unification problems of larger cardinality.

3.3. Reformulations

In this subsection, we consider reformulations of (some of) the notions introduced above from an order-theoretic, logical, algebraic, and category-theoretic point of view. This will shed a new light on the notions, and it allows us to utilize approaches and results from the respective areas in unification theory.

3.3.1. The order-theoretic point of view

Let E be an equational theory and Γ an E -unification problem with variables $\mathcal{X} := \text{Var}(\Gamma)$. We know that the relation $\leq_E^{\mathcal{X}}$ is a quasi-ordering on $\mathcal{U}_E(\Gamma)$ with associated equivalence relation $\doteq_E^{\mathcal{X}}$. Thus, $\leq_E^{\mathcal{X}}$ induces a partial ordering \leq on the set $U := \{[\sigma] \mid \sigma \in \mathcal{U}_E(\Gamma)\}$ of all $\doteq_E^{\mathcal{X}}$ -classes $[\sigma] := \{\tau \mid \sigma \doteq_E^{\mathcal{X}} \tau\}$:

$$[\sigma] \leq [\theta] \quad \text{iff} \quad \sigma \leq_E^{\mathcal{X}} \theta.$$

This allows us to investigate notions like complete and minimal complete sets of E -unifiers on the abstract order-theoretic level.

Thus, let (U, \leq) be an arbitrary partially ordered set. A subset C of U is called *complete* iff for all $u \in U$ there exists $c \in C$ such that $c \leq u$. A complete set C is called *minimal* iff it is minimal with respect to set inclusion.

3.10. LEMMA. *The complete set $C \subseteq U$ is minimal iff for all $x, y \in C$, $x \leq y$ implies $x = y$.*

PROOF. If the elements x, y of the complete set C satisfy $x < y$, then $C \setminus \{y\}$ is also complete, which shows that C is not minimal. Conversely, if C_1, C_2 are complete sets such that $C_1 \subset C_2$, then there exists $y \in C_2 \setminus C_1$. Since C_1 is complete, there exists $x \in C_1$ such that $x \leq y$, and since $y \notin C_1$, we have $x \neq y$. \square

The following lemma describes the connection between minimal complete sets and minimal elements in partially ordered sets.

3.11. LEMMA. *Let M be the set of \leq -minimal elements of U .*

1. *If $C \subseteq U$ is a minimal complete set, then $C = M$.*
2. *If M is complete, then it is minimal complete.*

PROOF. The second statement is obvious, since different \leq -minimal elements of U cannot be comparable w.r.t. \leq . To show the first statement, let $C \subseteq U$ be a minimal complete set. Obviously, $M \subseteq C$ since any \leq -minimal element must be contained in a complete set. To see the other inclusion, assume that $y \in C$ is not minimal. Thus, there exists an element $y' \in U$ such that $y' < y$. Since C is complete, there exists $x \in C$ such that $x \leq y'$. Thus, we have $x, y \in C$ with $x < y$, which shows that C cannot be minimal. \square

Figure 5 shows (the Hasse diagrams of) two partially ordered sets. The left one consists of an infinitely descending chain $x_1 > x_2 > x_3 > \dots$. Consequently, the set of \leq -minimal elements is empty, and thus not complete. The right one also contains an infinitely descending chain (consisting of the elements y_1, y_2, \dots), but the set of \leq -minimal elements (the elements z_1, z_2, \dots) is obviously complete. If

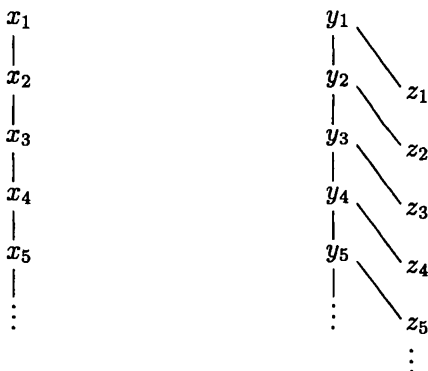


Figure 5: Two partially ordered sets.

$U = \{[\sigma] \mid \sigma \in \mathcal{U}_E(\Gamma)\}$ is the set of \doteq_E^χ -classes of E -unifiers of Γ , and \leq is the partial ordering on U induced by \leq_E^χ , then lemma 3.11 yields a nice characterization of all minimal complete sets of E -unifiers. If M is a subset of U , then a set of representatives of M is any subset of $\mathcal{U}_E(\Gamma)$ that contains for each class $m \in M$ exactly one representative, i.e., a unifier σ_m such that $[\sigma_m] = m$.

3.12. THEOREM. *Let M be the set of all \leq -minimal elements of U . If C is a minimal complete set of E -unifiers of Γ , then $M = \{[\sigma] \mid \sigma \in C\}$. Conversely, if M is complete, then any set of representatives of M is a minimal complete set of E -unifiers of Γ .*

As an immediate consequence of this theorem we can deduce

3.13. COROLLARY. *Let M be the set of all \leq -minimal elements of U .*

1. *A minimal complete set of E -unifiers of Γ exists iff M is complete.*
2. *If a minimal complete set of E -unifiers of Γ exists, then it is unique up to the equivalence $\stackrel{\mathcal{X}}{=}_E$.*

In [Baader 1989a], this order-theoretic point of view was used to derive different characterizations of unification type zero.

3.3.2. The algebraic and logical point of view

It is well known that the decision problems for elementary unification and for unification with constants correspond to natural classes of logical decision problems [Bockmayr 1992], and it turns out that the same is true for general unification.

Before stating these logical characterizations of E -unification, we recall some results from universal algebra about equationally defined classes (see, e.g., [Cohn 1965, Mal'cev 1971, Grätzer 1979] for more details). An equational theory E defines a *variety* (or equational class) $V(E)$, i.e., the class of all models of E . The theory E is called *non-trivial* if $V(E)$ contains algebras of cardinality > 1 , and *trivial* otherwise. Obviously, E is trivial iff $x =_E y$ for distinct variables x, y . If E is a non-trivial equational theory, then $V(E)$ contains free algebras over any set of generators. In fact, let $\mathcal{F}_0 := \text{Sig}(E)$, and let \mathcal{X} be a set of variables of cardinality α . Then the quotient term algebra $\mathcal{T}(\mathcal{F}_0, \mathcal{X}) / \stackrel{\mathcal{X}}{=}_E$ is a free algebra in $V(E)$. Its set of generators consists of the $=_E$ -classes of the variables, and this set has cardinality α since E was assumed to be non-trivial. We call this algebra the *E -free algebra with generators \mathcal{X}* .⁴ The fact that it is free in $V(E)$ means that any mapping from \mathcal{X} into an algebra $\mathcal{A} \in V(E)$ can uniquely be extended to a homomorphism of $\mathcal{T}(\mathcal{F}_0, \mathcal{X}) / \stackrel{\mathcal{X}}{=}_E$ into \mathcal{A} .

Now, we introduce the classes of formulae that correspond to equational unification problems. Let E be an equational theory, and $\mathcal{F}_0 := \text{Sig}(E)$ be the set of function symbols occurring in E . An *atomic \mathcal{F}_0 -formula* is an equation $s = t$. A *positive \mathcal{F}_0 -matrix* is built from atomic \mathcal{F}_0 -formulae using conjunction and disjunction. A *positive \mathcal{F}_0 -sentence* is a quantifier-prefix followed by a positive \mathcal{F}_0 -matrix that contains only variables introduced in the prefix. Without loss of generality we assume that the variables occurring in the prefix are all distinct. A *positive existential \mathcal{F}_0 -sentence* is a positive \mathcal{F}_0 -sentence whose prefix contains only existential quantifiers, and a *positive AE \mathcal{F}_0 -sentence* has a prefix consisting of a block of universal quantifiers, followed by a block of existential quantifiers. The positive (positive existential, positive AE) fragment of the equational theory E consists of the set of all positive (positive existential, positive AE) \mathcal{F}_0 -sentences that are valid in E , i.e., true in all models of E . Accordingly, for an \mathcal{F}_0 -algebra \mathcal{A} , the positive

⁴Strictly speaking, the generators are the $=_E$ -classes of the elements of \mathcal{X} , but since different variables belong to different classes, we slightly abuse the notation by identifying a variable $x \in \mathcal{X}$ with its $=_E$ -class.

(positive existential, positive AE) theory of \mathcal{A} is the set of all positive (positive existential, positive AE) \mathcal{F}_0 -sentences that are true in \mathcal{A} .

3.14. THEOREM. *Let E be a non-trivial equational theory, $\mathcal{F}_0 := \text{Sig}(E)$, and \mathcal{V} a countably infinite set of variables.*

1. *Elementary E -unification is decidable iff the positive existential fragment of E is decidable iff the positive existential theory of $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E$ is decidable.*
2. *E -unification with constants is decidable iff the positive AE fragment of E is decidable iff the positive AE theory of $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E$ is decidable.*

PROOF. (1.1) Let $\Gamma := \{s_1 =_E^? t_1, \dots, s_n =_E^? t_n\}$ be an elementary E -unification problem, and let $\text{Var}(\Gamma) = \{x_1, \dots, x_k\}$. The terms $s_1, t_1, \dots, s_n, t_n$ are \mathcal{F}_0 -terms with variables in $\text{Var}(\Gamma)$, which implies that

$$\phi_\Gamma := \exists x_1 \dots \exists x_k. s_1 = t_1 \wedge \dots \wedge s_n = t_n$$

is a positive existential \mathcal{F}_0 -sentence. We claim that Γ is E -unifiable iff ϕ_Γ holds in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E$ iff ϕ_Γ is valid in E .

Assume that σ is an E -unifier of Γ , i.e., $s_1\sigma =_E t_1\sigma, \dots, s_n\sigma =_E t_n\sigma$. Without loss of generality we may assume that σ introduces only variables from \mathcal{V} . Thus, the substitution σ may also be considered as a valuation of the variables $\{x_1, \dots, x_k\}$ by elements of $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E$. Conversely, any such valuation can be seen as a substitution. This shows that Γ is E -unifiable iff ϕ_Γ holds in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E$.

If ϕ_Γ is valid in all models of E , it obviously holds in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E \in V(E)$. Conversely, assume that ϕ_Γ holds in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E$. If ϕ_Γ is not valid in E , then there exists an algebra $\mathcal{A} \in V(E)$ in which ϕ_Γ does not hold. By the Löwenheim-Skolem theorem, we may without loss of generality assume that \mathcal{A} is countable. Thus, there exists a surjective homomorphism from $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E$ onto \mathcal{A} (extending an arbitrary surjection of \mathcal{X} onto the carrier of \mathcal{A}). Since validity of positive sentences is invariant under surjective homomorphisms,⁵ validity of ϕ_Γ in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E \in V(E)$ implies validity of ϕ_Γ in \mathcal{A} , which is a contradiction.

(1.2) Let $\phi = \exists x_1 \dots \exists x_n. \psi$ be a positive existential \mathcal{F}_0 -sentence. Without loss of generality we may assume that its matrix ψ is in disjunctive normal form, i.e., $\psi = \psi_1 \vee \dots \vee \psi_n$ where the formulae ψ_i are conjunctions of equations. Since existential quantifier distribute over disjunction, ϕ is valid in E (in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E$) iff one of the formulae $\exists x_1 \dots \exists x_n. \psi_i$ is valid in E (in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E$). Obviously, the formulae ψ_i can be translated into unification problems Γ_i , and as in part (1.1) of the proof we can show that Γ_i is unifiable iff $\exists x_1 \dots \exists x_n. \psi_i$ is valid in E (in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_E$).

(2) The second equivalence can be shown as in part (1.1) of the proof (since there we have only used the fact that ϕ_Γ is a positive \mathcal{F}_0 -sentence).

To see the first equivalence, assume that ϕ is a positive AE sentence. Skolemizing the universally quantified variables⁶ yields a positive existential $(\mathcal{F}_0 \cup \mathcal{F}_1)$ -sentence

⁵See [Mal'cev 1973], pp. 143, 144 for a proof.

⁶We must Skolemize the universally quantified variables since we are interested in validity instead of satisfiability.

ϕ' such that \mathcal{F}_1 is a set of constants (not contained in $\text{Sig}(E)$) and ϕ is valid in E iff ϕ' is valid in E . As in (1.2) of the proof, ϕ' can be translated into E -unification problems $\Gamma_{\psi'_i}$ such that ϕ' is valid in E iff one of these unification problems is unifiable. Obviously, the problems $\Gamma_{\psi'_i}$ are E -unification problem with constants since they contains the additional Skolem constants \mathcal{F}_1 . Conversely, any E -unification problem with constants can be turned into a positive AE sentence by replacing its free constants by universally quantified variables. \square

The reduction described in part (1.2) of the proof is exponential in the worst case since the disjunctive normal form of the matrix ψ can be exponential in the size of ψ . For syntactic equality (i.e., $E = \emptyset$), it can be shown that the problem of deciding validity of positive existential sentences is NP-complete, whereas the corresponding unification problem is linear [Kozen 1981].

Before we state the analogous correspondence between general E -unification and the (full) positive fragment of E , we introduce another class of unification problems, which turns out to be equivalent to general E -unification.

3.15. DEFINITION. An E -unification problem *with linear constant restrictions* (*lcr*) consists of an E -unification problem with constants, Γ , and a linear ordering $<$ on the variables and free constants occurring in Γ . A substitution σ is an *E -unifier* of $(\Gamma, <)$ iff it is an E -unifier of Γ that satisfies

$$x < c \text{ implies } c \text{ does not occur in } x\sigma$$

for all variables x and free constants c in Γ .

For example, the (syntactic) unification problem $\{f(x) = ? f(c)\}$ has $\{x \mapsto c\}$ as most general unifier. Under the restriction $x < c$, this unifier is not admissible.

3.16. THEOREM. Let E be a non-trivial equational theory, $\mathcal{F}_0 := \text{Sig}(E)$, and \mathcal{V} a countably infinite set of variables. Then the following statements are equivalent:

1. The positive theory of E is decidable.
2. The positive theory of $\mathcal{T}(\mathcal{F}_0, \mathcal{V}) / \equiv_E$ is decidable.
3. General E -unification is decidable.
4. E -unification with linear constant restrictions is decidable.

PROOF. We only give a sketch of the proof (see [Baader and Schulz 1996] for details).

In order to show (1) \Leftrightarrow (2), it is sufficient to show that a positive \mathcal{F}_0 -sentence ϕ is valid in E iff it is true in $\mathcal{T}(\mathcal{F}_0, \mathcal{V}) / \equiv_E$. This can be shown as in part (1.1) of the proof of theorem 3.14.

A given positive sentence ϕ can be turned into a positive existential sentence ϕ' by Skolemization. As in part (2) of the proof of theorem 3.14, validity of ϕ' can be reduced to validity of several E -unification problems, which are general since they may contain Skolem functions of arbitrary arity. This shows (3) \Rightarrow (1).

A given E -unification problem with linear constant restrictions $(\Gamma, <)$ can be transformed into a positive \mathcal{F}_0 -sentence $\phi_\Gamma^<$ as follows: the matrix of $\phi_\Gamma^<$ is simply

the conjunction of all equations in Γ . However, the constants in Γ are considered as variables in this matrix. The quantifier-prefix contains a universal quantifier for every free constant in Γ , and an existential quantifier for every variable in Γ . The order of the quantifiers is determined by the linear ordering $<$. It can be shown that $(\Gamma, <)$ is unifiable iff $\phi_\Gamma^<$ is valid in E . This proves (1) \Rightarrow (4).

Finally, (4) \Rightarrow (3) follows from the combination result in [Baader and Schulz 1996] (see section 6). \square

The following example, in which we assume $E = \{f(x) \approx f(x)\}$, illustrates the transformation of an E -unification problem with linear constant restrictions into a positive sentences, and of this positive sentence into a general E -unification problem (by Skolemization).

<i>unification with lcr</i>	<i>positive sentence</i>	<i>general unification</i>
$\{x =_E^? f(c)\}, x < c$	$\exists x. \forall y. x = f(y)$	$\{x =_E^? f(h(x))\}$
$\{x \doteq f(c)\}, c < x$	$\forall y. \exists x. x = f(y)$	$\{x =_E^? f(d)\}$

The problem $\{x =_E^? f(c)\}$ is not unifiable under the restriction $x < c$, since any unifier must replace x by $f(c)$, which contains the forbidden constant c . The corresponding positive sentence $\exists x. \forall y. x = f(y)$ is not valid since it says that f is a constant function, which is not true in all models of E . Finally, the general E -unification problem $\{x =_E^? f(h(x))\}$, which contains the Skolem function h , is not unifiable since one obtains an occurs check failure. Changing the linear ordering to $c < x$ leads to a unifiable unification problem with lcr, and the corresponding positive sentence is trivially valid.

3.3.3. The category-theoretic point of view

Let $\Gamma := \{s_i =_E^? t_i \mid i = 1, \dots, n\}$ be an E -unification problem over \mathcal{F} , and $\mathcal{X} := \text{Var}(\Gamma)$ be the finite set of variables occurring in Γ . Since all our calculations are done modulo E , we may consider the terms s_i and t_i as elements of $\mathcal{T}(\mathcal{F}, \mathcal{X})/_E$, the E -free algebra with generators \mathcal{X} . For example, let \mathcal{F} consist of a binary function symbol f , and let A axiomatize associativity of f , i.e., $A := \{f(x, f(y, z)) \approx f(f(x, y), z)\}$. The E -free algebra with generators \mathcal{X} is the free semigroup \mathcal{X}^+ , whose elements are the nonempty words over the alphabet \mathcal{X} . Instead of writing terms like $f(x, f(y, f(x, x)))$ in A -unification problems, we can omit the parentheses and all occurrences of the letter f , and simply write words like $xyxx$.

Also, since the instantiation quasi-ordering compares substitutions only on \mathcal{X} and modulo E , each substitution can be seen as a homomorphism from $\mathcal{T}(\mathcal{F}, \mathcal{X})/_E$ into an E -free algebra $\mathcal{T}(\mathcal{F}, \mathcal{Y})/_E$, where \mathcal{Y} is a suitable finite set (of variables or generators). For example, modulo A , the substitution $\sigma := \{x \mapsto f(x, f(y, f(x, x))), y \mapsto f(y, z)\}$ can be viewed as a homomorphism $\sigma: \{x, y\}^+ \rightarrow \{x, y, z\}^+$ that maps x to the word $xyxx$ and y to the word yz .

The E -unification problem Γ itself can be represented as a pair of homomorphisms between finitely generated E -free algebras. Indeed, let $\mathcal{I} := \{x_1, \dots, x_n\}$ be a set of cardinality n . If we define $\sigma, \tau: \mathcal{T}(\mathcal{F}, \mathcal{I}) / \equiv_E \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X}) / \equiv_E$ by

$$x_i \sigma := s_i \quad \text{and} \quad x_i \tau := t_i \quad (i = 1, \dots, n),$$

then $\delta: \mathcal{T}(\mathcal{F}, \mathcal{X}) / \equiv_E \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{Y}) / \equiv_E$ is an E -unifier of Γ iff $x_i \sigma \delta = s_i \delta = t_i \delta = x_i \tau \delta$,⁷ that is, iff $\sigma \delta = \tau \delta$. Consequently, any E -unification problem over \mathcal{F} can be represented as a parallel pair of morphisms in the following category:⁸

3.17. DEFINITION. Let E be an equational theory and \mathcal{F} be a signature such that $\text{Sig}(E) \subseteq \mathcal{F}$. The category $C_{\mathcal{F}}(E)$ is defined as follows:

1. The objects of $C_{\mathcal{F}}(E)$ are the finitely generated E -free algebras $\mathcal{T}(\mathcal{F}, \mathcal{X}) / \equiv_E$.
2. The morphisms of $C_{\mathcal{F}}(E)$ are the homomorphisms between these algebras. For a morphism $\delta: \mathcal{T}(\mathcal{F}, \mathcal{X}) / \equiv_E \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{Y}) / \equiv_E$, the algebra $\mathcal{T}(\mathcal{F}, \mathcal{X}) / \equiv_E$ is called its domain, and the algebra $\mathcal{T}(\mathcal{F}, \mathcal{Y}) / \equiv_E$ its codomain.
3. Composition $\sigma \delta$ of morphisms is the usual composition of mappings, which is only defined if the codomain of σ coincides with the domain of δ .

A *unification problem* in $C_{\mathcal{F}}(E)$ is a pair $\langle \sigma, \tau \rangle$ of morphisms $\sigma, \tau: \mathcal{T}(\mathcal{F}, \mathcal{I}) / \equiv_E \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X}) / \equiv_E$ having the same domain and the same codomain. A *unifier* of $\langle \sigma, \tau \rangle$ in $C_{\mathcal{F}}(E)$ is a morphism δ with domain $\mathcal{T}(\mathcal{F}, \mathcal{X}) / \equiv_E$ such that $\sigma \delta = \tau \delta$.

The instantiation quasi-order, and the notions complete and minimal complete set of unifiers as well as most general unifier can be adapted in an obvious way to this view of E -unification as a problem in $C_{\mathcal{F}}(E)$. For example, the morphism δ is a *most general* unifier of $\langle \sigma, \tau \rangle$ iff it is a unifier of $\langle \sigma, \tau \rangle$ such that, for all unifiers θ of $\langle \sigma, \tau \rangle$, there exists a morphism λ satisfying $\theta = \delta \lambda$.

Readers familiar with basic notions from category theory may have noticed that this definition of a most general unifier of $\langle \sigma, \tau \rangle$ strongly resembles the definition of a *coequalizer* of a parallel pair of morphisms (i.e., a pair with the same domain and the same codomain). The only difference is that for a most general unifier of $\langle \sigma, \tau \rangle$ to be a coequalizer, the morphism λ such that $\theta = \delta \lambda$ must always be *unique*.

It is easy to see that a most general unifier of $\langle \sigma, \tau \rangle$ need not be a coequalizer of this parallel pair. For example, the most general (syntactic) unifier $\delta := \{y \mapsto x\}$ of the equation $f(x, y) =^? f(y, x)$ can be viewed as a morphism $\delta_{\mathcal{Y}}: \mathcal{T}(\{f\}, \{x, y\}) \rightarrow \mathcal{T}(\{f\}, \mathcal{Y})$ for any finite set of variables \mathcal{Y} containing x . All these morphisms are most general unifiers of the parallel pair corresponding to the unification problem $f(x, y) =^? f(y, x)$, but only $\delta_{\{x\}}$ is a coequalizer. More generally, a most general unifier in $C_{\mathcal{F}}(\emptyset)$ need not be a coequalizer, but it can always be transformed into one by appropriately restricting the set of generators in its codomain.

For nonempty theories, such a transformation need not be possible, however. As shown in [Baader 1991], there exists an equational theory, namely the theory ACU

⁷Since terms are now viewed as elements of E -free algebras (i.e., \equiv_E -equivalence classes), we may write equality ($=$) in place of equality modulo E (\equiv_E).

⁸See [Pierce 1991] for basic definitions and results of category theory.

that axiomatizes an associative-commutative binary symbol f with a unit e , such that all solvable unification problems in $C_{\{f,e\}}(ACU)$ have a most general unifier, but not all solvable unification problems in this category have a coequalizer. In the applications of E -unification in automated deduction, the additional uniqueness requirement in the definition of a coequalizer is not relevant. Thus, one should stick with the definition of a most general unifier as introduced above, and not replace it by the one of a coequalizer.

As such, the simple observation that E -unification has a category-theoretic interpretation does not solve any problems: it just transforms them into a different representation. This new representation is only of interest if techniques and results from category theory can be used to solve new and interesting problems in unification theory. Rydeheard and Burstall [1985] use the category-theoretic representation of syntactic unification to derive a unification algorithm based on colimit constructions in $C_{\mathcal{F}}(\emptyset)$. In [Baader 1989b], results from category theory on so-called semi-additive categories are used to obtain results on unification modulo so-called commutative theories (see subsection 5.2 below).

Even though the construction of the category $C_{\mathcal{F}}(E)$ is quite natural, there are also other ways of representing unification problems in category-theoretic terms. Whereas Goguen [1989] just introduces the dual category of $C_{\mathcal{F}}(E)$ (where morphisms are inverse homomorphisms), Ghilardi [1997] takes a quite different approach: he considers the category of all algebras in $V(E)$ (not only the finitely generated free ones), and represents unification problems as finitely presented algebras in this category. In this setting, the proof that unification in Boolean algebras and in primal algebras is unitary [Nipkow 1990] becomes trivial.

3.4. Survey of results for specific theories

Research in unification theory has produced results on unification properties of a great variety of equational theories. In this section, we will briefly review some of these results, with an emphasis on the more recent ones that are not yet covered by previous surveys of the area [Siekmann 1989, Jouannaud and Kirchner 1991, Kapur and Narendran 1992a, Baader and Siekmann 1994]. For each theory, we are interested in the decision problem and its complexity as well as its unification type and the existence of unification algorithms and procedures. Depending on which kind of unification problems (elementary, with constants, or general) is considered, there may exist different results for a given theory.

Associativity

The theory $A_f := \{f(f(x, y), z) \approx f(x, f(y, z))\}$ axiomatizes associativity of the binary function symbol f .

Decision problem: This problem, which is very hard and had been open for a long time, was finally solved by Makanin [1977], who proves decidability of A_f -unification with constants (see also [Pécuchet 1981, Jaffar 1990, Abdulrab and Pécuchet 1989, Schulz 1993]). Using general combination techniques and an

extension of Makanin's algorithm [Schulz 1992], decidability of general A_f -unification was shown in [Baader and Schulz 1992, Baader and Schulz 1996]. The decision problem for A_f -unification is NP-hard [Benanav, Kapur and Narendran 1985]. The known upper bound is still higher, even though there has recently been considerable progress in lowering the bound: the 3-NEXPTIME result by Koscielski and Pacholski [1990] was first improved to EXPSPACE by Gutiérrez [1998], then to NEXPTIME by Plandowski [1999a], and finally to PSPACE [Plandowski 1999b]. Interestingly, the last two results no longer need Makanin's algorithm, i.e., they yield a new decision procedure that is independent of Makanin's result.

Unification type: infinitary for all three kinds of unification problems [Plotkin 1972] (see also example 3.7).

Unification procedures: Plotkin [1972] describes a minimal unification procedure for general A_f -unification, which can even deal with several associative function symbols. In general, this procedure does not yield a decision procedure since it need not terminate even for non-solvable problems or problems having a finite minimal complete set of A_f -unifiers. For certain restricted types of A_f -unification problems, modifications of Plotkin's procedure can be turned into decision procedures that are simpler than Makanin's general procedure [Auffray and Enjalbert 1992, Schmidt 1998].

Commutativity

The theory $C_f := \{f(x, y) \approx f(y, x)\}$, which axiomatizes commutativity of the binary function symbol f , has already been considered in example 3.6.

Decision problem: NP-complete for C_f -unification with constants and general C_f -unification. The hardness result for unification with constants is mentioned in [Garey and Johnson 1979], where it is attributed to Sethi (private communication, 1977). A simple NP-hardness proof due to Narendran (private communication, 1993) is sketched in [Baader and Siekmann 1994]. It is easy to see that this proof can also be used to show NP-hardness of elementary C_f -unification (private communication by Narendran, 1997).⁹ NP-decision procedures for general C_f -unification can easily be obtained from the simple unification algorithm sketched in example 3.6: instead of testing all possible sets Γ' , the non-deterministic decision procedure first guesses such a set Γ' , and then tests whether this set has a syntactic unifier.

Unification type: finitary for all three kinds of unification problems [Siekmann 1979].

Unification algorithms: In addition to Siekmann's simple (non-minimal) unification algorithm for general C_f -unification [Siekmann 1979], various other methods have been proposed [Fages 1983, Kirchner 1985, Herold 1987]. However, none of them directly produces a *minimal* complete set of C_f -unifiers.

⁹In this proof, simply replace the constants a, b by the terms $t_a := f(x, f(x, x))$ and $t_b := f(x, x)$ and add for each propositional variable q an equation $f(x_q, y_q) =_{C_f} f(t_a, t_b)$, which makes sure that x_q is instantiated either by t_a or by t_b .

Distributivity

The theories $D_{f,g}^l := \{f(x, g(y, z)) \approx g(f(x, y), f(x, z))\}$ and $D_{f,g}^r := \{f(g(y, z), x) \approx g(f(y, x), f(z, x))\}$ axiomatize left-distributivity and right-distributivity of f over g , and their union $D_{f,g} := D_{f,g}^l \cup D_{f,g}^r$ axiomatizes (both-sided) distributivity of f over g . In addition, we consider combinations of these theories with A_g and $U_f := \{f(x, e) \approx x, f(e, x) \approx x\}$.

Decision problem: $D_{f,g}^l$ -unification (and, by symmetry, $D_{f,g}^r$ -unification) with constants is decidable in polynomial time [Tidén and Arnborg 1987].

If one adds a unit for f , i.e., considers $D_{f,g}^l \cup U_f$ (or $D_{f,g}^r \cup U_f$), then the problem becomes much harder since A_f -unification can be reduced to $(D_{f,g}^l \cup U_f)$ -unification. Decidability of $(D_{f,g}^l \cup U_f)$ -unification with constants was shown in [Schmidt-Schauß 1996b]. Since this decision procedure can be extended to cope with linear constant restrictions, general results on the combination of decision procedures [Baader and Schulz 1996] imply that general $(D_{f,g}^l \cup U_f)$ -unification is decidable.

For unification modulo both-sided distributivity, the decision problem was open for quite a while. After some preliminary decidability results for restricted classes of $D_{f,g}$ -unification problems [Contejean 1993, Schmidt-Schauß 1992], decidability of $D_{f,g}$ -unification with constants was finally shown by Schmidt-Schauß [1996a]. His non-deterministic algorithm reduces solvability of $D_{f,g}$ -unification problems with constants to A_f -unification with constants and ACU -unification with linear constant restrictions. Thus, the algorithm is of quite high complexity, compared to the best known lower bound, which is NP-hard [Tidén and Arnborg 1987].

Undecidability of $(D_{f,g} \cup A_g)$ -unification with constants was proved in [Szabó 1982, Siekmann and Szabó 1989]. This negative result has been strengthened in [Tidén and Arnborg 1987]: every equational theory that lies above $(D_{f,g} \cup A_g)$ or $(D_{f,g}^l \cup U_f \cup A_g)$ and is consistent with Peano arithmetic (where f stands for multiplication, g for addition, and e for 1) has an undecidable unification problem. Decidability of $(D_{f,g} \cup U_f)$ -unification is still an open problem.

Unification type: infinitary for $D_{f,g}$ -unification problems with constants and general $D_{f,g}$ -unification problems. Szabó [1982] gives an example of a $D_{f,g}$ -unification problem with constants whose minimal complete set of unifiers is infinite. The existence of minimal complete sets of $D_{f,g}$ -unifiers (for all three kinds of unification problems) is a consequence of the fact that the $=_{D_{f,g}}$ -class of a given term is always finite [Szabó 1982], which implies that the instantiation quasi-ordering $\leq_{D_{f,g}}^X$ is Noetherian [Szabó 1982, Bürckert et al. 1989].

$D_{f,g}^l$ -unification (and, by symmetry, $D_{f,g}^r$ -unification) with constants is unitary, and an mgu can be computed in polynomial time [Tidén and Arnborg 1987].

Associativity-commutativity

The theories $AC_f := A_f \cup C_f$ and $ACU_f := AC_f \cup U_f$ will be considered in more detail in subsection 5.1. Examples of operations satisfying these identities are addition and multiplication of (rational, real, etc.) numbers.

Decision problem: NP-complete for unification problems with constants and general unification problems both for AC_f and ACU_f [Kapur and Narendran 1992a]. Elementary ACU_f -unification problems always have a trivial solution, and solvability of elementary AC_f -unification problems is decidable in polynomial time using linear programming [Domenjoud 1991].

Unification type: ACU_f is unitary for elementary and finitary for the two other kinds of unification problems, and AC_f is finitary for all three kinds of unification problems [Livesey and Siekmann 1975, Stickel 1981, Pages 1987]. The number of unifiers in a minimal complete set of AC_f -unifiers may be doubly-exponential in the size of a given elementary AC_f -unification problem [Kapur and Narendran 1992b].

Unification algorithms: Because unification modulo associativity-commutativity has many applications in automated deduction, a great variety of unification algorithms has been developed for AC_f and ACU_f [Stickel 1975, Livesey and Siekmann 1975, Kirchner 1985, Fortenbacher 1985, Büttner 1986a, Herold 1987, Herold and Siekmann 1987, Lincoln and Christian 1989, Boudet, Contejean and Devie 1990] (see also subsection 5.1).

Associativity-commutativity-idempotency

We consider the theories $ACI_f := AC_f \cup \{f(x, x) \approx x\}$, its extension by a unit e , $ACUI_f := ACI_f \cup U_f$, and by a zero n , $ACUZI_f := ACUI_f \cup \{f(x, n) \approx n\}$. Examples of operations satisfying these identities are union and intersection of sets. The theory $ACUI_f$ will be considered in more detail in subsection 5.1.

Decision problem: For all three theories, the decision problem is polynomial for elementary unification and for unification with constants, and NP-complete for general unification [Kapur and Narendran 1992a, Narendran 1996b]. Like syntactic unification, ACI_f - and $ACUI_f$ -unification with constants are not only in P , but even P -complete [Hermann and Kolaitis 1997].

Unification type: $ACUI_f$ is unitary for elementary and finitary for the two other kinds of unification problems, and ACI_f is finitary for all three kinds of unification problems [Livesey and Siekmann 1975, Büttner 1986b, Baader and Büttner 1988, Kapur and Narendran 1992b]. As with AC_f , the number of ACI_f -unifiers in a minimal complete set may be doubly-exponential in the size of a given elementary ACI_f -unification problem [Kapur and Narendran 1992b]. Hermann and Kolaitis show that computing the cardinality of a minimal complete set of unifiers for given ACI_f - or $ACUI_f$ -unification problems is $\#P$ -hard, which implies that this function cannot be computed in polynomial time, unless $P = NP$ [Hermann and Kolaitis 1997].

Unification algorithms: Baader and Büttner [1988] describe an algorithm for $ACUI_f$ -unification problems with constants consisting of a single equation, and Kapur and Narendran [1992b] sketch an algorithm for general ACI_f -unification.

Abelian groups

The theory of Abelian groups is defined by the identities $AG_f := ACU_f \cup \{f(i(x), x) \approx e\}$.

Decision problem: trivial for elementary unification, polynomial for unification with constants [Baader and Siekmann 1994], and NP-complete for general unification [Schulz 1997].

Unification type: unitary for elementary unification and for unification with constants [Lankford, Butler and Brady 1984], and finitary for general unification [Schmidt-Schauß 1989b, Boudet, Jouannaud and Schmidt-Schauß 1989]. Computing the cardinality of a minimal complete set of unifiers for a given general AG_f -unification is again $\#P$ -hard [Hermann and Kolaitis 1996].

Unification algorithms: Lankford et al. [1984] describe an algorithm for AG_f -unification with constants, and Schmidt-Schauß [1989b] shows that this algorithm can be combined with an algorithm for syntactic unification into an algorithm for general AG_f -unification.

Commutative and Boolean rings

Let CRU denote the well-known axioms for commutative rings with a (multiplicative) unit, and BR the theory of Boolean rings.

Decision problem: As sketched in [Baader and Siekmann 1994], undecidability of elementary CRU -unification is an easy consequence of the fact that Hilbert's 10th problem is undecidable [Matiyasevich 1971, Davis 1973].

For the theory BR , the decision problem is NP-complete for elementary unification, Π_2^P -complete for unification with constants, and PSPACE-complete for general unification [Baader 1998].

Unification type: The unification type of CRU is at least infinitary, even for elementary unification [Burris and Lawrence 1990].¹⁰

BR is unitary for elementary unification and for unification with constants [Büttner and Simonis 1987, Martin and Nipkow 1989b, Martin and Nipkow 1989a], and finitary for general unification [Schmidt-Schauß 1989b]. As with the theory of Abelian groups, the problem of computing the cardinality of a minimal complete set of unifiers is $\#P$ -hard for general BR -unification [Hermann and Kolaitis 1996].

Unification algorithms: Algorithms that compute most general unifiers for elementary BR -unification and BR -unification with constants are described in [Büttner and Simonis 1987, Martin and Nipkow 1989b, Martin and Nipkow 1989a]. General combination methods can be used to obtain algorithms for general BR -unification [Schmidt-Schauß 1989b, Boudet et al. 1989].

Endomorphisms

The theory $End_{h,g} := \{h(g(x, y)) \approx g(h(x), h(y))\}$ states that the unary function symbol h behaves like an endomorphism for the binary function symbol g , and

¹⁰The closely related theory of commutative *semirings* is known to be of unification type zero w.r.t. elementary unification [Franzen 1992]

$End_{h,e} := \{h(e) \approx e\}$ states that h behaves like an endomorphism for the constant symbol e . We consider these two theories in combination with some of the theories introduced above:

Decision problem: Solvability of $End_{h,g}$ -unification problems with constants is decidable [Vogel 1978].

For the theories $End_{h,g} \cup AC_g$ and $End_{h,g} \cup End_{h,e} \cup ACU_g$, solvability of unification problems with constants is undecidable [Narendran 1996a].

In contrast, solvability of unification problems with constants is decidable for the theory $End_{h,g} \cup End_{h,e} \cup ACUI_g$. In [Baader and Narendran 1998] it is shown that this problem is EXPTIME-complete.

A similar result holds for $End_{h,g} \cup ACUI_g$: for this theory, the decision problem is known to be co-NP-hard and in EXPTIME [Guo, Narendran and Shukla 1998].

Finally, for $End_{h,g} \cup End_{h,e} \cup AG_g$, decidability of unification with constants was shown in [Baader 1993]. Since this decidability result can be extended to unification with linear constant restrictions, general combination results yield decidability for general unification modulo this theory [Baader and Nutt 1996].

Unification type: The theory $End_{h,g}$ is unitary for unification with constants [Vogel 1978].

$End_{h,g} \cup End_{h,e} \cup ACU_g$ and $End_{h,g} \cup End_{h,e} \cup ACUI_g$ are of type zero, even for elementary unification [Baader 1993, Baader 1989b].

$End_{h,g} \cup End_{h,e} \cup AG_g$ is unitary for elementary unification and for unification with constants [Nutt 1990, Baader 1993], and finitary for general unification [Baader and Nutt 1996].

In addition to investigating unification properties of specific equational theories of interest, unification theory also tries to develop more general methods, and thus to obtain results for whole classes of equational theories. Since unification modulo equational theories is in general undecidable (as illustrated by some of the examples above), and also unification properties such as the unification type of a given theory are in general undecidable [Nutt 1991], approaches that apply to all equational theories are likely to yield very weak results. For example, the general E -unification procedure introduced in section 4.1, which can be used to enumerate a complete set of E -unifiers, is very inefficient, and usually does not yield a decision procedure or a (minimal) E -unification algorithm even for unitary or finitary theories whose unification problem is decidable. In order to obtain more useful results, one can try to develop methods that work for appropriately restricted classes of theories. There are basically two different ways of introducing appropriate restrictions on equational theories. *Syntactic approaches* impose restrictions on the syntactic form of the identities defining the equational theories. The unification methods produced by these approaches are usually also of a quite syntactic nature: as with the rule-based approach to syntactic unification, they transform the given unification problem into a problem in solved form (section 4). In contrast, *semantic approaches* depend on properties of the (free) algebras defined by the equational theory. Unification problems are translated into equations over certain algebraic structures, which (in some cases) can be solved using known results from mathematics (section 5).

4. Syntactic methods for E -unification

In this section we discuss two syntactic approaches to generating complete sets of E -unifiers, using inference systems extending the set \mathcal{U} presented in section 2.2.3. We first consider the general problem (E -unification in arbitrary theories) and show how it can be solved by adding a single rule to introduce identities into the transformation process; this simple method is proved to be complete and some restrictions which preserve completeness are discussed. We then present the most significant special case of the general problem, when the equational theory can be presented by a convergent set of rewrite rules. This method, called *narrowing*, has been thoroughly investigated, and we will present the major results in the framework of transformation rules.

4.1. E -unification in arbitrary theories

In this section, we present a rule for introducing identities into inference steps in \mathcal{U} in such a way that a complete set of E -unifiers for an arbitrary set E of equations may be generated. By specializing various aspects of the resultant calculus (and its completeness proof), we will obtain more practical methods for the special case of convergent sets of rewrite rules. The results of this section are based on [Gallier and Snyder 1989, Snyder 1991].

In this section we assume that the reader is familiar with the basic concepts of rewriting (especially equational proofs, reduction orderings, ground convergence, and critical pairs) discussed in [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook). By *rewrite proof* we refer to a sequence of rewrite steps between two terms of the form

$$s \xrightarrow{*} u \xleftarrow{*} t$$

where u is in normal form. We will use $e[u]$ in the following to represent a equation (or identity) with a distinguished occurrence of a subterm u in one of its terms; in such a context $e[r]$ will denote the result of replacing this subterm with the term r . We will use systems $P; S$, representing unification problems and sets of equations in solved form, as before.

4.1. DEFINITION. For any equational theory E , a substitution θ is an E -*solution* (or simply a *solution* when E is understood) of a system $P; S$ if it is an E -unifier of every equation in P , and a unifier of every equation in S .

4.1.1. The calculus \mathcal{G}

The set \mathcal{G} of inference rules consists of the rules Trivial, Decomposition, Orientation, and Variable Elimination from \mathcal{U} , plus the following rule for introducing identities:

Lazy Paramodulation (LP):

$$\{e[u]\} \cup P; S \implies_{lp} \{l \stackrel{?}{=} u, e[r]\} \cup P; S$$

for a fresh variant¹¹ of the identity $l \approx r$ from $E \cup E^{-1}$, and where (i) u is *not* a variable, and (ii) if l is not a variable, then the top symbols of l and u are identical, and no other inference rule may be applied to the equation $l =^? u$ before it is subjected to a Decomposition step.

Computation in \mathcal{G} proceeds as in \mathcal{U} , starting with an initial system of the form $\{s =^? t\}; \emptyset$ and applying inference rules in an attempt to find some terminal system $\emptyset; S$ representing an E -unifier σ_S of s and t . Clearly, by the general characteristics of E -unification discussed above, such a process can not share the nice properties of \mathcal{U} which we discussed in section 2.2.4. However, it is possible to say quite a lot about how to restrict the application of rules, as we shall see.

4.1.2. Completeness of \mathcal{G}

It can be shown easily that the calculus \mathcal{G} is *sound* in the sense that a solution it produces is always an E -unifier; however this proof does not give much insight into the properties of \mathcal{G} and we refer the interested reader to [Gallier and Snyder 1989]. It is more interesting to consider the issue of completeness, which is considerably more complex than in the standard case. What we want to show is that if we consider the (finitely-branching but infinite) search tree of every possible transformation sequence starting from $\{s =^? t\}; \emptyset$, then the leaves form a complete set of E -unifiers for s and t . However, it is simpler to state and prove this in the following “non-deterministic” form.

4.2. THEOREM. *Let E be a non-trivial equational theory and P be a set of unification problems. If θ is an E -solution of $P; \emptyset$, then there exists a sequence*

$$P; \emptyset \xRightarrow{*} \emptyset; S$$

(with S in solved form) in the calculus \mathcal{G} such that $\sigma_S \leq_E^{\mathcal{X}} \theta$, where $\mathcal{X} = \text{Vars}(P)$.

There are three main stages to the proof. First we will prove the result given certain strong restrictions on the equational theory E . Then we construct a kind of “abstract completion” of E which has the requisite restrictions; finally, we show that any transformation sequence using this abstract completion can be converted into one using simply E .

The major difficulty in proving completeness of equational inference systems is generally in dealing with the restriction that equational steps not take place at variable positions (hence, “ u is not a variable” in LP). The solution, due to Peterson [1983], is to work with a restricted form of substitution in the proof.

4.3. DEFINITION. Given a rewrite system R , a substitution θ is *R -reduced* (or just *reduced* if R is unimportant) if for every $x \in \text{Dom}(\theta)$, $x\theta$ is in R -normal form.

¹¹By a *fresh variant* we refer to an expression that has been renamed with fresh variables that do not occur anywhere else in the previous computation. Whenever we mention a rewrite rule or identity used in an inference step, we will assume that it has been so renamed.

Note that it is always possible for any θ and terminating set of rules R to find an R -equivalent reduced substitution θ' . This allows us to assume, when “lifting” rewrite steps at the ground level to inference steps, that the position is a non-variable.

Another essential ingredient in our proof is the notion of an “oriented ground instance” of an identity.

4.4. DEFINITION. Let E be a non-trivial equational theory and \succ be a reduction ordering total on ground terms. The set of *ground instances* of E is

$$Gr(E) := \{ l\rho \approx r\rho \mid l\rho \text{ and } r\rho \text{ are ground and } l \approx r \in E \cup E^{-1} \}.$$

The set of *oriented ground instances* of E is

$$Gr^\succ(E) := \{ l\rho \longrightarrow r\rho \mid l\rho \approx r\rho \in Gr(E) \text{ and } l\rho \succ r\rho \}.$$

A member $l\rho \longrightarrow r\rho$ of such a set is called *reduced* if ρ is reduced with respect to the entire set.¹² For any E , the set of reduced oriented ground instances is denoted R_E .

An important fact about $Gr(E)$ is the following.

4.5. PROPOSITION. *For any two ground terms s and t , there exists an equational proof $s \xrightarrow{*}_E t$ iff there exists a proof $s \xrightarrow{*}_{Gr(E)} t$*

This is easily proved by showing that equational steps are closed under instantiation, and hence we can instantiate any “unbound variables” by ground terms so that only ground instances of identities from E are used.

Another kind of restriction on proofs, which will be essential in proving the “no inferences into variable positions” restriction in our completeness result, is the subject of the next definition and lemma.

4.6. DEFINITION. Let $u\theta$ be an instance of u , and R a set of rewrite rules. A rewrite step $u\theta \longrightarrow_R u'$ is *based on u* iff the redex is at a non-variable position in u (equivalently, is not wholly contained within a term introduced by θ). A rewrite sequence $s\theta \xrightarrow{*}_R t$ is *based on s* (or simply *basic*) iff either $s\theta = t$ (reflexive case) or it starts with a rewrite step based on s , e.g.,

$$s\theta \longrightarrow_R (s\theta)[r\rho] = s[r]\theta\rho \xrightarrow{*}_R t$$

and the remainder is based on $s[r]$. A rewrite proof $s\theta \xrightarrow{*} \longleftarrow^* t\theta$ is *basic* if the left side is based on s and the right side is based on t .

Intuitively, this means that no rewrite step can take place at a term introduced by any substitution.

The relationship between reduced substitutions, reduced oriented ground instances, ground convergence, and basic rewrite sequences is now explored.

¹²This notion is well-defined, as it could more formally be defined by induction on a suitable ordering of rules, using the fact that l can not be a variable when E is non-trivial.

4.7. LEMMA. *Let E be a non-trivial equational theory such that $Gr^\succ(E)$ is ground convergent, and $s\theta$ be a ground term such that θ is R_E -reduced. Then for any rewrite sequence $s\theta \xrightarrow{*} t$ using rules from $Gr^\succ(E)$ to reduce $s\theta$ to its normal form t , there exists a basic rewrite sequence $s\theta \xrightarrow{*} t$ using rules only from R_E .*

PROOF. Since $Gr^\succ(E)$ is ground canonical, we may choose any fair strategy for reduction; in particular, we may specify that at each step, among all the possible rules that could be used for reduction, we choose one that is minimal in the lexicographic extension of \succ to pairs of terms. But then for any $l\rho \rightarrow r\rho$ used in the sequence, ρ must be reduced, or else the rule would not be minimal. Thus, there exists a rewrite sequence from $s\theta$ to t using rules only from R_E ; clearly, since all substitutions involved are reduced, this is also a basic sequence. \square

For our purposes we may summarize these results as follows.

4.8. COROLLARY. *Let E be an equational theory such that $Gr^\succ(E)$ is ground convergent. For any ground terms $s\theta$ and $t\theta$, where θ is reduced with respect to $Gr^\succ(E)$, the following are equivalent:*

1. *$s\theta$ and $t\theta$ are E -equivalent.*
2. *There exists a basic rewrite proof for $s\theta$ and $t\theta$ using rules from $Gr^\succ(E)$.*

We now prove our completeness result in the special case we have been discussing.

4.9. LEMMA. *Let E be a non-trivial equational theory such that $Gr^\succ(E)$ is ground convergent, and P be a set of unification problems. If θ is a $Gr^\succ(E)$ -reduced solution of $P; \emptyset$, then there exists a sequence*

$$P; \emptyset \xrightarrow{*} \emptyset; S$$

(with S in solved form) in the calculus \mathcal{G} such that $\sigma_S \leq^X \theta$ for $X = \text{Vars}(P)$.

PROOF. We proceed by induction, using the following measure. The complexity of a system $P; S$ and its solution θ is a four-tuple $\langle m, n_1, n_2, n_3 \rangle$, where

- m = The total number of rewrite steps in all the minimal-length basic rewrite proofs for equations in $P\theta$;
- n_1 = The number of distinct variables occurring in equations $u = ? v \in P$ such that $u\theta = v\theta$ and $u\theta$ is in $Gr^\succ(E)$ -normal form;
- n_2 = The number of symbols occurring in equations $u = ? v \in P$ such that $u\theta = v\theta$ and $u\theta$ is in normal form;
- n_3 = The number of equations in P of the form $t = ? x$, where t is not a variable, and such that $t\theta = x\theta$ and $t\theta$ is in normal form.

The associated (well-founded) ordering is the lexicographic ordering using the natural ordering on positive integers.

We show by induction on this measure that if θ is a solution of a system $P; S'$, with S' in solved form, there exists a transformation sequence

$$P; S' \xrightarrow{*} \emptyset; S$$

where $\sigma_S \leq^{\mathcal{X}} \theta$ for $\mathcal{X} = \text{Vars}(P, S')$.

The base case of the induction consists of a system $\emptyset; S$ and the result is trivial, since *a fortiori* $\sigma_S \leq \theta$. For the induction step, suppose $P = \{u =^? v\} \cup P'$. If $u\theta = v\theta$ with $u\theta$ in normal form; then we proceed as before with the inference system \mathcal{U} to generate a transformation step to a smaller system containing the same set of variables, and with the same solution (cf. lemma 2.4). As with \mathcal{U} , any equation introduced into S must keep this set in solved form. Completing this with the induction hypothesis, we have

$$P; S' \Rightarrow_{\mathcal{U}} P''; S'' \xRightarrow{*} \emptyset; S$$

such that $\sigma_S \leq^{\mathcal{X}} \theta$ with $\mathcal{X} = \text{Vars}(P, S')$.

Otherwise, without loss of generality, pick a rewrite step from the term $u\theta$ in a minimal-length basic rewrite proof $u\theta \rightarrow \xrightarrow{*} \xleftarrow{*} v\theta$, in which a reduced ground instance $l\rho \rightarrow r\rho$ was used. If we let $\theta' = \theta\rho$, then this first step was in fact $u[u']\theta' = u[l]\theta' \rightarrow u[r]\theta'$, where u' can not be a variable (since θ is reduced). In addition, the top symbols of u' and l are identical if l is not a variable. Hence, there exists some transformation step

$$\{u[u'] =^? v\} \cup P'; S' \Rightarrow_{l\rho} \{l =^? u', u[r] =^? v\} \cup P'; S'$$

to a new system which has a smaller complexity with respect to its new solution θ' . (It also contains additional variables, i.e., those in $\text{Vars}(l, r)$). By the induction hypothesis we can continue this with:

$$\{l =^? u', u[r] =^? v\} \cup P'; S' \xRightarrow{*} \emptyset; S$$

such that $\sigma_S \leq^{\mathcal{X}} \theta'$ with $\mathcal{X} = \text{Vars}(l, r, P, S')$. But, since $x\theta = x\theta'$ for every $x \in \text{Vars}(P, S')$, we are done. \square

The second stage of our main completeness proof for \mathcal{G} involves constructing a set of identities fitting the conditions of the previous lemma. We do this by a kind of abstract completion of E :

4.10. DEFINITION. Let $Cr(E)$ be the set of critical pairs w.r.t. \succ of E , created from fresh variants of identities in E using the inference system \mathcal{U} to calculate the requisite *mgus*. Then, for each $i \geq 0$, define

$$\begin{aligned} E^0 &= E \\ &\vdots \\ E^{i+1} &= E^i \cup Cr(E^i) \\ &\vdots \\ E^\omega &= \bigcup_{n \geq 0} E^n \end{aligned}$$

The entire point of this construction is contained in the following lemma, which can be proved using techniques familiar from [Dershowitz and Plaisted 2001], Chapter 9 of this Handbook (for a specific proof, see Theorem 6.1.7 in [Snyder 1991]).

4.11. LEMMA. *For any E , $Gr^\succ(E^\omega)$ is ground convergent and equivalent to E on ground terms.*

Thus, we can (conceptually, at least) use E^ω to construct transformation sequences as just shown in lemma 4.9. The second main lemma of our completeness proof for \mathcal{G} shows how to convert such a transformation sequence into one using only identities from E .

4.12. LEMMA. *For any sequence*

$$P; \emptyset \xRightarrow{*} \emptyset; S$$

introducing identities from E^ω , and such that σ_S is an E -unifier for P , there exists a sequence

$$P; \emptyset \xRightarrow{*} \emptyset; S'$$

introducing identities only from E , such that $S \subseteq S'$ and $x\sigma_{S'} = x\sigma_S$ for every $x \in \text{Vars}(P)$.

PROOF. The basic idea is to use the calculus \mathcal{G} itself to construct critical pairs. The complexity measure in our inductive proof is as follows. The *depth* of an identity $e \in E^\omega$ is the least k such that $e \in E^k$; the complexity of a transformation sequence is the (finite) multiset of the depths of all identities from E^ω introduced, with the associated (well-founded) multiset ordering.

The base case being trivial, we proceed directly to the induction step. Suppose the transformation sequence uses some identity $r_1\sigma \approx l_1[r_2]\sigma$ of non-zero depth, obtained by forming a critical pair from $l_1[l'] \approx r_1$ and $l_2 \approx r_2$ (each of smaller depth) with $\sigma = \text{mgu}(l', l_2)$. We show how the original use of the critical pair in a LP step can be simulated by two LP steps involving the component identities, plus some number of \mathcal{U} -transformations to simulate the construction of the critical pair. There are two cases, depending on which direction the critical pair was used in.

Case One. Suppose the critical pair was $r_1\sigma \approx l_1[r_2]\sigma$, e.g.,

$$\begin{aligned} &\xRightarrow{*} \{e[u]\} \cup P; S' \\ &\xRightarrow{\text{lp}} \{r_1\sigma \stackrel{?}{=} u, e[l_1[r_2]\sigma]\} \cup P; S' \\ &\xRightarrow{*} \emptyset; S \end{aligned}$$

where an additional Decomposition is possibly applied afterwards to $r_1\sigma \stackrel{?}{=} u$ (if $r_1\sigma$ is not a variable). This sequence can be converted into:

$$\begin{aligned} &\xRightarrow{*} \{e[u]\} \cup P; S' \\ &\xRightarrow{\text{lp}} \{r_1 \stackrel{?}{=} u, e[l_1[l']]\} \cup P; S' \\ &\xRightarrow{\text{lp}} \{l_2 \stackrel{?}{=} l'_1, r_1 \stackrel{?}{=} u, e[l_1[r_2]]\} \cup P; S' \\ &\xRightarrow{*} \{r_1\sigma \stackrel{?}{=} u, e[l_1[r_2]\sigma]\} \cup P; S \cup [\sigma] \\ &\xRightarrow{*} \emptyset; S \cup [\sigma'] \end{aligned}$$

(where by $[\sigma]$ we mean a set of equations representing the bindings in σ). This sequence has a smaller complexity, as it replaced a critical pair by two identities of strictly smaller depth. The second line from the bottom represents the calculation of the *mgu*; these bindings apply only to terms from the two equations, although as they are carried along in the solution set they may change as the result of additional substitutions (hence the change to σ'). The (possible) Decomposition step after the first LP step in the original is delayed until after the computation of σ .

Case Two. Suppose the critical pair was $l_1[r_2]\sigma \approx r_1\sigma$; in this case, we may assume that the overlap in this critical pair is not at the root, since otherwise we could apply case one. Our original sequence is thus:

$$\begin{aligned} &\xrightarrow{*} \{e[u]\} \cup P; S' \\ &\xRightarrow{\text{lp}} \{l_1[r_2]\sigma =^? u, e[r_1\sigma]\} \cup P; S' \\ &\xrightarrow{*} \emptyset; S \end{aligned}$$

where Decomposition is applied to $l_1[r_1]\sigma \approx u$ at some point after the LP step (since l_1 has at least one function symbol above the overlap position). This sequence becomes:

$$\begin{aligned} &\xrightarrow{*} \{e[u]\} \cup P; S' \\ &\xRightarrow{\text{lp}} \{l_1[l'_1] =^? u, e[r_1]\} \cup P; S' \\ &\xRightarrow{\text{lp}} \{l_2 =^? l'_1, l_1[r_2] =^? u, e[r_1]\} \cup P; S' \\ &\xrightarrow{*} \{l_1[r_2]\sigma =^? u, e[r_1\sigma]\} \cup P; S \cup [\sigma] \\ &\xrightarrow{*} \emptyset; S \cup [\sigma'] \end{aligned}$$

The Decomposition step is delayed until after the computation of σ . This sequence is, again, of smaller complexity than the original.

Note in both cases that the variables in $\text{Dom}(\sigma)$ are (effectively) fresh, as they occur in the component identities but not in the critical pair; thus, $x\sigma_{S'} = x\sigma_S$ for all $x \in \text{Vars}(P)$ as required. \square

We may now present the proof of our main completeness result.

Proof of theorem 4.2. First, note that we may assume that $P\theta$ contains only ground equations, using a straight-forward Skolemization argument (viz. [Snyder 1991], p.90). If θ is an E -unifier of P , we may construct an $\text{Gr}^\gamma(E)$ -reduced substitution θ' such that $\theta =_E \theta'$. We then apply lemma 4.9, using rules from E^ω , to obtain a sequence

$$P; \emptyset \xrightarrow{*} \emptyset; S$$

where $\sigma_S \leq^{\mathcal{X}} \theta'$ for $\mathcal{X} = \text{Vars}(P)$. This is then converted, using the technique of lemma 4.12 to a new sequence using rules only from E :

$$P; \emptyset \xrightarrow{*} \emptyset; S'$$

where $x\sigma_S = x\sigma_{S'}$ for every $x \in \text{Vars}(P)$. Thus, we may conclude that $\sigma_S \leq^{\mathcal{X}}_E \theta$, where $\mathcal{X} = \text{Vars}(P)$, as required. \square

4.2. Restrictions on E -unification in arbitrary theories

In this section we describe two refinements of the calculus \mathcal{G} that have been suggested:

- The restriction on a equation $l = ? u$ introduced by LP, when l is not a variable, that the top symbol of l and u must be the same, can be strengthened so that the entire overlap of the non-variable positions in the two terms must be identical.
- The restriction in LP that u not be a variable may be strengthened so that u can not even be a term introduced into P by substitution (i.e., Variable Elimination) at any point in the sequence.

Both of these restrictions in some sense extend the original restrictions on \mathcal{G} *hereditarily*, in the first case inheriting the restriction on top symbols down into the terms, and in the second, inheriting the non-variable restriction throughout the history of the equation, and regarding terms introduced by variable elimination as being second-class citizens which do not play a direct role in equational inferences, but only serve to constrain the application of rules. This is called the *basic* restriction, as it rests on the existence of basic rewrite proofs as shown above.

For lack of space, we do not consider these refinements to \mathcal{G} in detail here, although the second will form an essential part of the calculus in the next section. For the first, see [Dougherty and Johann 1992], and also [Socher-Ambrosius 1994] (where a further refinement is presented); for the second see [Moser 1993].

4.3. Narrowing

In this section we consider the most important special case of the E -unification problem, when the equational theory can be represented by a ground convergent set of rewrite rules. In this case, the conversion of transformation sequences to simulate critical pair generation is not necessary, and we can take a closer look at the completeness proof and the restrictions that can be imposed on the calculus. In particular, we shall from the start consider the existence of basic rewrite proofs as fundamental, and develop a new representation for problems which prevents LP inferences at terms introduced by substitutions.

A *constraint system* (or simply *system* in the rest of the section) is either the symbol \perp (representing failure) or a triple consisting of a multiset P of equations (representing the schema of the problem, in a sense that will become clear below), a set C of equations (representing constraints on variables in P), and a set S of equations (representing bindings in the solution). The set C plays a role similar to the multiset P in section 2.2.4, and rules from \mathcal{U} will be applied to $C; S$ as before. The equational problems being worked on are in fact $P\sigma_S$, the separation into the schema P and constraints $C; S$ serving to enforce the *basic* restriction on the application of LP mentioned above. As expected, a substitution θ is said to be a solution (or E -unifier) of a system $P; C; S$ if it E -unifies each equation in P , and unifies each of the equations in C and S ; the system \perp has no E -unifiers.

We assume that our rewrite system R (representing E) is ground convergent with respect to a reduction ordering \succ , and consists of a numbered sequence of rules

$$\{l_1 \longrightarrow r_1, l_2 \longrightarrow r_2, \dots, l_n \longrightarrow r_n\}.$$

The *index* of a rule will be its number in this sequence, and will be used in a certain refinement of our inference system.

4.3.1. The calculus \mathcal{B}

In this section we present the rules which are used in the calculus \mathcal{B} for *basic narrowing*. We will first consider a simple set of rules and prove its completeness, and then consider refinements and modifications based on the details of the proof.

The set \mathcal{B} consists of the following six rules.

Trivial:

$$P; \{s \stackrel{?}{=} s\} \cup C'; S \implies P; C'; S$$

Decomposition:

$$P; \{f(s_1, \dots, s_n) \stackrel{?}{=} f(t_1, \dots, t_n)\} \cup C'; S \implies P; \{s_1 \stackrel{?}{=} t_1, \dots, s_n \stackrel{?}{=} t_n\} \cup C'; S$$

Orient:

$$P; \{t \stackrel{?}{=} x\} \cup C'; S \implies P; \{x \stackrel{?}{=} t\} \cup C'; S$$

if t is not a variable.

Basic Variable Elimination:

$$P; \{x \stackrel{?}{=} t\} \cup C'; S \implies P; C' \{x \mapsto t\}; S \{x \mapsto t\} \cup \{x \approx t\}$$

if x does not occur in t . (Note that the substitution is *not* applied to the set P .)

(Modulo the changes to Variable Elimination, these are just the non-failure rules from \mathcal{U} , adapted for constraint systems; we shall denote these first four rules as \mathcal{S} .)

Constrain:

$$\{e\} \cup P'; C; S \implies_{\text{con}} P'; \{e\sigma_S\} \cup C; S$$

Lazy Paramodulation:

$$\{e[u]\} \cup P; C; S \implies_{\text{lp}} \{e[r]\} \cup P; \{l\sigma_s \stackrel{?}{=} u\sigma_s\} \cup C; S$$

(with the exact same restrictions as given above in section 4.1.1).

Essentially, this calculus is no different from \mathcal{G} , except that it is designed to enforce the basic restriction, by separating out the parts of terms that were introduced into the problem by substitution (i.e., Variable Elimination) and those that were not (the "schema"). The latter constitute the only positions where equational inferences may take place in the basic strategy. The completeness proof is hence very similar to lemma 4.9. We will add more restrictions to the way that certain choices are made, however, which will give us the ability to restrict our calculus correspondingly.

4.13. THEOREM. *Let R be a ground convergent set of rewrite rules. If θ is an R -solution of $P; \emptyset; \emptyset$, then there exists a sequence*

$$P; \emptyset; \emptyset \xRightarrow{*}_B \emptyset; \emptyset; S$$

such that $\sigma_S \leq_R^{\mathcal{X}} \theta$, where $\mathcal{X} = \text{Vars}(P)$.

PROOF. As in our completeness proof for \mathcal{G} , we may assume that $P\theta$ is ground and that θ is R -reduced, since the relation \leq_R does not distinguish between R -equivalent substitutions. Thus, we will prove a stronger result, that when θ is R -reduced, then in fact $\sigma_S \leq^{\mathcal{X}} \theta$.

The complexity of a system $P; C; S$ and associated solution θ is $\langle M, n_1, n_2, n_3 \rangle$, where

- M = The multiset of all terms occurring in $P\theta$;
- n_1 = The number of distinct variables in C ;
- n_2 = The number of symbols in C ;
- n_3 = The number of equations in C of the form $t = ? x$, where t is not a variable.

The associated ordering is the lexicographic ordering using the multiset extension of the reduction ordering \succ for the first component, and the ordering on natural numbers for the remaining components.

Our induction shows that if θ is a solution of a system $P; C; S'$, with S' in solved form, there exists a transformation sequence

$$P; C; S' \xRightarrow{*} \emptyset; \emptyset; S$$

where $\sigma_S \leq^{\mathcal{X}} \theta$, where $\mathcal{X} = \text{Vars}(P, C, S')$.

The base case $\emptyset; \emptyset; S$ is again trivial. For the induction step, there are several overlapping cases.

(1) If $C = \{u = ? v\} \cup C'$, then $u\theta = v\theta$ and we use S to generate a transformation step to a smaller system containing the same set of variables, and with the same solution (cf. lemma 2.4). Completing this with the induction hypothesis, we have

$$P; C; S' \Rightarrow_S P''; C'; S'' \xRightarrow{*} \emptyset; \emptyset; S$$

such that $\sigma_S \leq^{\mathcal{X}} \theta$ for $\mathcal{X} = \text{Vars}(P, C, S')$.

(2) If $P = \{u =^? v\} \cup P'$ and $u\theta = v\theta$, then we may apply *Constrain* to obtain a smaller system (reducing the component M) with the same solution and the same set of variables, and we conclude as in the previous case.

(3) Suppose $P = \{u =^? v\} \cup P'$ and there is some redex in either $u\theta$ or $v\theta$; without loss of generality, assume the former. We may also assume that the redex is innermost, and that if more than one instance of a rule from R reduces this redex, we choose the rule $l\rho \rightarrow r\rho$ with the smallest index in the set R . Note that, since θ is R -reduced, the redex must occur inside the non-variable positions of u ; thus we have the following transformation:

$$\{u[u'] =^? v\} \cup P'; C; S' \Rightarrow_{l\rho} \{u[r] =^? v\} \cup P'; \{l\sigma_{s'} =^? u'\sigma_{s'}\} \cup C; S'$$

to a system which is smaller with respect to its new solution $\theta' = \theta\rho$ (since the new equation introduced into C is an identity modulo θ'). Note that θ' is still R -reduced. By the induction hypothesis we have

$$\{u[r] =^? v\} \cup P'; \{l\sigma_{s'} =^? u'\sigma_{s'}\} \cup C; S' \xRightarrow{*} \emptyset; \emptyset; S$$

such that $\sigma_S \leq^{\mathcal{X}} \theta'$ with $\mathcal{X} = \text{Vars}(l, r, P, C, S')$, and since $x\theta = x\theta'$ for every $x \in \text{Vars}(P, C, S')$, the induction is complete. \square

4.3.2. Standard narrowing

An interesting feature of this proof is that it also provides for the completeness of an alternate (and historically earlier) version of narrowing due to Fay [1979], which does not distinguish between substitution positions and other positions in the problem.

Let us define the calculus \mathcal{N} for *standard narrowing* as the inference system \mathcal{B} with the following change: Basic Variable Elimination is replaced by the following transformation:

Variable Elimination:

$$P; \{x =^? t\} \cup C'; S \Rightarrow P\{x \mapsto t\}; C'\{x \mapsto t\}; S\{x \mapsto t\} \cup \{x \approx t\}$$

if x does not occur in t .

(The *Constrain* rule might also be changed so that it does not instantiate an equation when moving it from P to C , however, since σ_S is always idempotent, the existing rule would have the same effect.)

The only difference is that the set P is kept instantiated with the substitution defined by S during the transformation process, so that substitution positions can be used for narrowing.

4.14. COROLLARY. *Let R be a ground convergent set of rewrite rules. If θ is an R -solution of $P; \emptyset; \emptyset$, then there exists a sequence*

$$P; \emptyset; \emptyset \xRightarrow{*}_{\mathcal{N}} \emptyset; \emptyset; S$$

in the calculus \mathcal{N} such that $\sigma_S \leq^{\mathcal{X}} \theta$ with $\mathcal{X} = \text{Vars}(P)$.

The proof is essentially the same as the previous one, since the same transformation sequence can be used in each case.

The difference between the two inference systems is that \mathcal{B} restricts the application of inference rules to a smaller set of positions than \mathcal{N} does, and hence the search tree for solutions is narrower.

4.4. Strategies and refinements of basic narrowing

There is a variety of strategies and refinements that can be developed for the basic narrowing calculus without destroying completeness. Most of these, in one way or another, can be derived from a close examination of the completeness proof just given. In this section we briefly describe the most important of these.

4.4.1. Composite rules for basic narrowing

The first observation that can be made is that it is not necessary to consider all possible sequences of transformation rules, since we either solve (standard) unification problems (e.g., equations between two identical terms in $P\theta$) or simulate rewriting at the ground level by unifying left-hand sides of rules with non-variable positions in terms, at the non-ground level. Thus, we may use the following two composite rules as an alternate form of \mathcal{B} :

Solve (\Rightarrow_{sol}):

$$\{e\} \cup P'; C; S \Rightarrow_{\text{con}} P'; \{e\sigma_S\} \cup C; S \xRightarrow{*}_S P'; C\eta; S\eta \cup [\eta]$$

(i.e., $\eta = \text{mgu}(e\sigma_S)$).

Narrow (\Rightarrow_{nar}):

$$\{e[u]\} \cup P; C; S \Rightarrow_{\text{lp}} \{e[r]\} \cup P; \{l\sigma_S \stackrel{?}{=} u\sigma_S\} \cup C; S \xRightarrow{*}_S \{e[r]\} \cup P; C\eta; S\eta \cup [\eta]$$

(that is, $\eta = \text{mgu}(l\sigma_S, u\sigma_S)$), where $l \rightarrow r$ is a fresh variant from R .

The completeness proof goes through with few changes. Note that in this formulation, no new equations remain in C after each step. A similar set of composite rules could be given for \mathcal{N} .

4.4.2. Simplification

The inference rules in \mathcal{S} (like \mathcal{U}) are significant in that they can be applied whenever we want during a transformation sequence without affecting the outcome; in our inductive proof, we may observe that they make the problem smaller without changing the solution. Such rules are extremely important in reducing the search space for a solution.

4.15. DEFINITION. A transformation \gg is called a *simplification rule* for \mathcal{B} if whenever $P; C; S \gg P'; C'; S'$, then θ is an R -reduced solution of $P'; C'; S'$ iff $\theta|_{\text{Vars}(P, C, S)}$ is an R -reduced solution to $P; C; S$, and $P'; C'; S'$ is smaller in the induction ordering used in Theorem 4.13 with respect to θ than $P; C; S$ w.r.t. $\theta|_{\text{Vars}(P, C, S)}$.

The restrictions in this definition ensure that such a rule can be used any time it applies in the induction step to obtain a smaller system without changing the solution (w.r.t. the variables in the left side).

Thus, the rules in \mathcal{S} are simplification rules in this respect. There are many other ad-hoc simplification rules that have been suggested for narrowing. For example, we may perform a form of Decomposition within P when we know that this does not remove a redex.

Problem Decomposition:

$$\{f(s_1, \dots, s_n \stackrel{?}{=} f(t_1, \dots, t_n))\} \cup P'; C'; S \implies \{s_1 \stackrel{?}{=} t_1, \dots, s_n \stackrel{?}{=} t_n\} \cup P'; C; S$$

if the symbol f does not occur at the top of the left-side of a rule in R .

In the induction in the completeness proof this rule decreases the measure (specifically, it reduces the component M). Clearly it does not change the set of solutions. Therefore, we may apply this rule any time, in any context, without affecting the completeness properties of the calculus.

Such rules can be applied “eagerly” to produce smaller problems, hopefully reducing the search space.

4.16. DEFINITION. If \mathcal{T} is a subset of rules for some calculus \mathcal{C} , then the *eager \mathcal{T} strategy* requires that a rule from $\mathcal{C} \setminus \mathcal{T}$ may only be applied if no rule from \mathcal{T} applies anywhere in the system.

Simplification rules can be performed eagerly.

4.17. THEOREM. Let R be a ground convergent set of rewrite rules, and \mathcal{A} be a set of simplification rules. If θ is an R -solution of $P; \emptyset; \emptyset$, then there exists a sequence

$$P; \emptyset; \emptyset \xRightarrow{*}_{B \cup \mathcal{A}} \emptyset; \emptyset; S$$

under the eager \mathcal{A} strategy such that $\sigma_S \leq^{\mathcal{X}}_R \theta$, where $\mathcal{X} = \text{Vars}(P)$.

The proof proceeds as before, with the exception that in the induction step, we must use a simplification step if one applies; as noted above, the conditions of a simplification rule ensure that the induction in the completeness proof goes through.

One of the most useful simplification rules is reducing the problem set by the set of rules R . From an abstract point of view, we may motivate such equational inferences as follows. If $u\theta \xrightarrow{*}_{E} v\theta$ and $u' \xrightarrow{*}_{E} u$, then, since equational proofs are closed under instantiation, we have $u'\theta \xrightarrow{*}_{E} u\theta \xrightarrow{*}_{E} v\theta$. Thus, we can not change the set

of solutions by performing equational inferences on the problem terms themselves, for example, by reducing them.

From the point of view of our calculus, we might observe that in the rule Narrow just introduced, if no application of Variable Elimination is ever applied to a variable from the system on the left side, then the set of solutions is unchanged by this transformation: the substitution generated must in this case apply only to l and r , and hence we have, at the ground level, replaced $e[u]\theta\rho = e[l]\theta\rho = e[l\rho]\theta$ with $e[r\rho]\theta$. Since the properties of θ were not involved, this means that effectively we have done a rewrite step $u[l\rho] \rightarrow_R u[r\rho]$. Alternately, we might say that if you end up doing Variable Elimination on $x \stackrel{?}{=} t$ for $x \in \text{Dom}(\theta)$ for some solution θ , then you are assuming that $x\theta = t\theta$; this cuts down on the number of possible solutions.

The resultant rule is:

Reduce (\Rightarrow_{red}):

$$\begin{aligned} \{e[u]\} \cup P; C; S &\Rightarrow_{\text{ip}} \{e[r]\} \cup P; \{l \stackrel{?}{=} u\sigma_S\} \cup C; S \\ &\stackrel{*}{\Rightarrow} \{e[r\rho]\} \cup P; C; S \cup [p] \end{aligned}$$

where $l \rightarrow r$ is a fresh variant from R (note that the variables in $\text{Dom}(\rho)$ occur only in r), and where the last line involves only Trivial, Decomposition, and Variable Elimination applied to the variables from l (i.e., $l\rho = u$).

Note that in the context of \mathcal{B} , we are losing some “basicness” by instantiating fully the right-hand side r ; below we shall consider how to recover some of the basic restriction lost in this fashion.

4.18. PROPOSITION. *The Eager Reduce Strategy is complete for \mathcal{B} and \mathcal{N} .*

Historically, the narrowing calculus was the first to be invented, by Fay [1979]; the basic narrowing calculus was developed by Hullot [1980], and it was observed by Réty [1987] that reduction needed to be modified in this setting. A study of basic narrowing with reduction, to which our treatment is heavily indebted, may be found in [Nutt, Réty and Smolka 1989]. In the next two sections we present further refinements which may also be found in [Bockmayr, Krischer and Werner 1992] and [Nutt et al. 1989]. For a comprehensive study of basic inference systems, the reader is referred to [Bachmair, Ganzinger, Lynch and Snyder 1995] and to [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook).

4.4.3. Redex orderings and variable abstraction

One of the useful properties of convergent systems mentioned above is that any strategy which can find a redex in a reducible term is sufficient for reducing terms to normal form, and hence for generating rewrite proofs. For example, at the ground level we might always look for redices in depth-first, left-to-right order. More generally, we may define a *redex ordering* \prec_{red} as an ordering on the positions in an equation which contains the proper subterm ordering (i.e., for any $u[u']$ with $u \neq u'$,

we have $u' \prec_{red} u$). Before considering whether a term t is reducible at a position π by some rule, we must consider all positions $\pi' \prec_{red} \pi$. The completeness proof could be sharpened by such an ordering simply by adding that we must choose the minimal redex according to the redex ordering (such a redex must be innermost). In such a case, the positions less than this redex may be assumed to be irreducible. No further narrowing steps need be performed at such positions, and in fact, we could remove these parts of the term and move them into the solved part of the system to enforce this.

Variable Abstraction (\Rightarrow_{abst}):

$$\{e[s]\} \cup P; C; S \Rightarrow \{e[x]\} \cup P; \{x \stackrel{?}{=} s\} \cup C; S$$

if x is a fresh variable.

A new version of the narrowing rule could then be presented which abstracts out terms which are known to be reduced.

Redex Ordered Narrow (\Rightarrow_{ron}):

$$\begin{aligned} \{e[u]\} \cup P; C; S &\Rightarrow_{lp} \{e[r]\} \cup P; \{l\sigma_s \stackrel{?}{=} u\sigma_s\} \cup C; S \xRightarrow{*}_S \{e[r]\} \cup P; C\eta; S\eta \cup [\eta] \\ &\xRightarrow{*}_{abst} \{e'[r]\} \cup P; C\eta \cup C'; S\eta \cup [\eta] \end{aligned}$$

where u occurs at position π in e , and Variable Abstraction is applied eagerly to all positions $\pi' \prec_{red} \pi$ in e to obtain e' .

The substitution of this version of Narrow in \mathcal{N} preserves completeness; the fundamental idea is that whenever a term (at the ground level in our completeness proof) may be assumed to be reduced, it may be moved into the constraint part of the system without losing completeness. This leads to a further use for Variable Abstraction in propagating what is known about reduced terms: if a term occurs in S , then (at the ground level) it may be assumed to be reduced, and hence other occurrences of this term may be abstracted out.

Propagation:

$$\{e[u]\} \cup P'; C; \{x \approx t[s]\} \cup S \Rightarrow_{prop} \{e[y]\} \cup P'; C; \{x \approx t[s], y \approx s\} \cup S$$

if $u\sigma_S = s$ is a non-variable and y is a fresh variable.

This rule is a simplification rule if we change the complexity measure in the proof to

$$\langle M, i, n_1, n_2, n_3 \rangle$$

where the additional component i is the number of non-variable symbols occurring in P . Clearly it changes the solution θ of a system to a new solution $\theta\{y \mapsto s\theta\}$ which satisfies the condition for a simplification rule.

Returning to our Reduce rule, we observe that in the context of \mathcal{B} , Reduce may instantiate terms into r that are known to be reduced; Propagation can remove these again. The combination of Reduction with Eager Propagation effectively gives us the more complex form of “basic simplification” described for example in [Bachmair et al. 1995] and [Nutt et al. 1989], see also [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook).

4.4.4. Failure rules

Unlike our presentation of the calculus \mathcal{U} , we have chosen here not to present failure rules from the outset, in order to highlight the essential issues first. The conditions under which sequences may fail are of two kinds. First, the failure rules for \mathcal{U} (Symbol Clash and Occur Check) may be applied to the sets C and S as before, since these represent unification problems; however, in this case the corresponding Solve, Narrow, or Reduce would simply not be performed.

The second class of conditions basically amount to checking for violations of the reducibility conditions in a system. At the ground level during the completeness proof, the substitution θ is kept reduced, and in addition, certain assumptions can be made about the existence of redices in terms. However, we have to be careful, as our proof only allows us to assume that all substitutions are R -reduced, and that no redex may be reduced below its root, or at the root by an equation of lower index.

This leads to the following rule:

Blocking ($\Rightarrow_{\text{block}}$):

$$P; C; S \Rightarrow \perp$$

if some term in S is R -reducible, or if some term in C is reducible below the root.

The Eager Blocking Strategy is complete, since the completeness proof requires the converse of the condition of this rule at all times. Note that this rule could be applied in the middle of a composite rule, for example, just after moving the equation into the set C in Narrow.

In order to account for reduction at the top of equations in C , it is preferable to add a further restriction to our Narrowing rule:

Narrow (\Rightarrow_{nar}):

$$\{e[u]\} \cup P; C; S \Rightarrow_{\text{lp}} \{e[r]\} \cup P; \{l\sigma_s \stackrel{?}{=} u\sigma_s\} \cup C; S \xRightarrow{*}_S \{e[r]\} \cup P; C\eta; S\eta \cup [\eta]$$

where $l \rightarrow r$ is a fresh variant from R and $l\sigma_s\eta$ is not the instance of the left-side of any rule of lower index from R .

This rule is consistent with Redex Orderings.

5. Semantic approaches to E -unification

The syntactic approaches to E -unification introduced above can be seen as extensions of the rule-based approach to syntactic unification, which use the identities defining the equational theory E to come up with additional transformation rules. In contrast, semantic approaches to E -unification try to utilize algebraic properties of the models of the equational theories. The two most prominent instances of the approach are

1. Unification in Boolean algebras and rings [Büttner and Simonis 1987, Martin and Nipkow 1989b, Martin and Nipkow 1989a], and its generalization to finite and to primal algebras [Büttner 1988, Büttner, Estenfeld, Schmid, Schneider and Tidén 1990, Nipkow 1990, Kirchner and Ringeissen 1994], and
2. Unification modulo the theories *ACU*, *ACUI*, and *AG* (see subsection 3.4 for references to result on unification modulo these theories).

In the following, we concentrate on the approach used in the second case since it can be generalized to a whole class of equational theories, called commutative theories in [Baader 1989b] and monoidal theories in [Nutt 1990]. For such theories, unification can be reduced to solving linear equations in a corresponding semiring.¹³ In the following, we introduce the class of commutative/monoidal theories, show how the corresponding semiring is defined, and how unification in commutative/monoidal theories can be reduced to solving linear equations in this semiring. In contrast to the syntactic approaches introduced above, general unification problems cannot be solved directly by the semantic approach described below. However, for commutative/monoidal theories, the known techniques for combining unification algorithms can always be used to extend an algorithm for unification with constants to an algorithm for general unification [Baader and Nutt 1996].

The theories

$$\begin{aligned}
 ACU &:= \{f(x, y) \approx f(y, x), f(f(x, y), z) \approx f(x, f(y, z)), f(x, e) \approx x\}, \\
 ACUI &:= ACU \cup \{f(x, x) \approx x\}, \\
 AG &:= ACU \cup \{f(x, i(x)) \approx e\}
 \end{aligned}$$

will be used as examples throughout this section. The introduction of the class of commutative/monoidal theories was motivated by the observation that the known algorithms for unification modulo these three theories have many common features.

5.1. Unification modulo *ACU*, *ACUI*, and *AG*: an example

We will first restrict our attention to elementary unification, and then show how the methods can be extended to unification with constants.

Elementary unification

To illustrate how the algorithms for elementary unification modulo these three theories work, let us consider the problem of unifying the two terms $f(x, f(x, y))$ and $f(z, f(z, z))$.

Let us start with the theory *ACU*. Obviously, the substitution $\sigma_1 := \{x \mapsto z_1, y \mapsto z_1, z \mapsto z_1\}$ is a syntactic unifier of this pair of terms, and thus also an *ACU*-unifier of $\Gamma_{ACU} := \{f(x, f(x, y)) \stackrel{?}{=}_{ACU} f(z, f(z, z))\}$. There are, however, *ACU*-unifiers of Γ_{ACU} that are not syntactic unifiers of the two terms: $\sigma_2 := \{x \mapsto$

¹³A semiring is similar to a ring, with the only difference being that its addition is just required to form an Abelian monoid, and not necessarily an Abelian group.

$e, y \mapsto f(z_2, f(z_2, z_2)), z \mapsto z_2\}$ is an example of such a unifier, and $\sigma_3 := \{x \mapsto f(z_3, f(z_3, z_3)), y \mapsto e, z \mapsto f(z_3, z_3)\}$ is another one. None of these substitutions is a most general *ACU*-unifier of Γ_{ACU} , but their "combination"

$$\begin{aligned}\sigma &:= \{x \mapsto f(x\sigma_1, f(x\sigma_2, x\sigma_3)), y \mapsto f(y\sigma_1, f(y\sigma_2, y\sigma_3)), \\ &\quad z \mapsto f(z\sigma_1, f(z\sigma_2, z\sigma_3))\} \\ &=_{ACU} \{x \mapsto f(z_1, f(z_3, f(z_3, z_3))), y \mapsto f(z_1, f(z_2, f(z_2, z_2))), \\ &\quad z \mapsto f(z_1, f(z_2, f(z_3, z_3)))\}\end{aligned}$$

is. For example, σ_2 can be obtained as an *ACU*-instance of σ by applying the substitution $\{z_1 \mapsto e, z_3 \mapsto e\}$. More generally, any finite collection $\sigma_1, \dots, \sigma_n$ of *ACU*-unifiers of a given *ACU*-unification problem can be combined in this way to a new *ACU*-unifier σ , which has all the unifiers σ_i as *ACU*-instances. In our example, there still remains the question of how we have found the three unifiers $\sigma_1, \sigma_2, \sigma_3$, and why their combination is a most general *ACU*-unifier of the problem.

In order to explain how we came up with these unifiers, assume that τ is an *ACU*-unifier of Γ_{ACU} , and that z' is a variable introduced by τ , i.e., z' occurs in (at least) one of the terms $x\tau, y\tau, z\tau$. It is easy to see that $f(x, f(x, y))\tau =_{ACU} f(z, f(z, z))\tau$ implies that the number of occurrences of z' in $f(x, f(x, y))\tau$ coincides with the number of occurrences of z' in $f(z, f(z, z))\tau$. Thus, if $|x\tau|_{z'}, |y\tau|_{z'}, |z\tau|_{z'}$ respectively denote the number of occurrences of z' in $x\tau, y\tau, z\tau$, then we have $2|x\tau|_{z'} + |y\tau|_{z'} = 3|z\tau|_{z'}$, i.e., the numbers $|x\tau|_{z'}, |y\tau|_{z'}, |z\tau|_{z'}$ are nonnegative integer solutions of the linear equation

$$2x + y = 3z.$$

Thus, every variable introduced by an *ACU*-unifier of a given *ACU*-unification problem yields a non-trivial¹⁴ solution of the linear equation corresponding to the problem in the semiring of all nonnegative integers (with addition and multiplication as semiring operations). For the unifier σ introduced above, the variable z_1 yields the solution (1, 1, 1), z_2 yields (0, 3, 1), and z_3 yields (3, 0, 2). What makes these three solutions special is that they are the minimal non-trivial solutions of $2x + y = 3z$ (w.r.t. the component-wise \leq -ordering on triples). Consequently, any solution can be obtained as a (nonnegative) linear combination of these three solutions.

Conversely, a substitution that introduces only variables (or free constants) corresponding to solutions of the linear equation is an *ACU*-unifier of the corresponding *ACU*-unification problem. For example, the substitution $\tau := \{x \mapsto f(z'f(z''), f(z'', z'')), y \mapsto f(z', f(z', f(z', z'))), z \mapsto f(z', f(z'f(z''), z''))\}$ is an *ACU*-unifier of Γ_{ACU} since $2 \cdot 1 + 4 = 3 \cdot 2$ and $2 \cdot 3 + 0 = 3 \cdot 2$. The solutions (1, 4, 2) and (3, 0, 2) can be obtained as linear combination of the minimal solutions:

$$\begin{aligned}(1, 4, 2) &= 1 \cdot (1, 1, 1) + 1 \cdot (0, 3, 1) + 0 \cdot (3, 0, 2), \\ (3, 0, 2) &= 0 \cdot (1, 1, 1) + 0 \cdot (0, 3, 1) + 1 \cdot (3, 0, 2).\end{aligned}$$

¹⁴Variables not introduced by the unifier correspond to the trivial solution (0, ..., 0).

This fact can be used to obtain a substitution λ such that $u\tau =_{ACU} u\sigma\lambda$ for all $u \in \{x, y, z\}$: $\lambda := \{z_1 \mapsto z', z_2 \mapsto z', z_3 \mapsto z''\}$.

To sum up, we have seen that a given elementary *ACU*-unification problem corresponds to a system¹⁵ of linear equations, which must be solved in the semiring \mathcal{N} of all nonnegative integers. A most general *ACU*-unifier of the problem is obtained by combining the unifiers corresponding to the (finitely many) minimal solutions of the system of linear equations. The important property of the set of minimal solutions is that it generates all solutions as linear combinations in \mathcal{N} . The fact that this set is always finite is an easy consequence of Dickson's Lemma [Dickson 1913]. Methods for computing this set can, for example, be found in [Huet and Lang 1978, Lambert 1987, Clausen and Fortenbacher 1989, Boudet et al. 1990, Pottier 1991, Domenjoud 1991, Contejean and Devie 1994, Filgueira and Tomás 1995].

The theory *ACUI* can be treated similarly, with the only difference being that the semiring \mathcal{N} must be replaced by the Boolean semiring \mathcal{BS} , which consists of the truth values 0 and 1, and has conjunction as its multiplication and disjunction as its addition operation. In fact, modulo *ACUI* it is no longer necessary that the *numbers* of occurrences of variables on the left-hand side and the right-hand side of the equation coincide. It is sufficient that each variable that occurs on the right-hand side also occurs on the left-hand side and vice versa. Thus, the linear equation corresponding to the *ACUI*-unification problem $\Gamma_{ACUI} := \{f(x, f(x, y)) =_{ACUI}^? f(z, f(z, z))\}$ is $x + y = z$, and it is easy to see that all solutions in \mathcal{BS} can be generated as linear combinations in \mathcal{BS} of the solutions (1, 0, 1) and (0, 1, 1). The most general *ACUI*-unifier obtained from this generating set of solutions is $\sigma' := \{x \mapsto z_1, y \mapsto z_2, z \mapsto f(z_1, z_2)\}$. The *ACU*-unifier σ_1 from above is also an *ACUI*-unifier of Γ_{ACUI} , and it can be obtained as an *ACUI*-instance of σ' via the substitution $\lambda' := \{z_1 \mapsto z_1, z_2 \mapsto z_1\}$. Since the Boolean semiring \mathcal{BS} is finite, there always exists a *finite* set of solutions that generates all solutions as linear combinations in \mathcal{BS} .

For the theory *AG*, the presence of the inverse operation leads to the fact that both the coefficients and the solutions of the linear equations corresponding to an *AG*-unification problem may also be negative integers. Thus, the semiring to be considered here is in fact a ring, namely the ring \mathcal{Z} of all integers. The linear equation corresponding to the *AG*-unification problem $\Gamma_{AG} := \{f(x, f(x, y)) =_{AG}^? f(z, f(z, z))\}$ coincides with the one obtained from Γ_{ACU} , but in \mathcal{Z} there exists a smaller set generating all solutions, consisting of (0, 3, 1) and (1, -2, 0). Thus, the substitution $\sigma'' := \{x \mapsto z_2, y \mapsto f(z_1, f(z_1, f(z_1, f(i(z_2), i(z_2))))), z \mapsto z_1\}$ is a most general *AG*-unifier of Γ_{AG} . General methods for computing such a finite generating set of solutions of systems of linear equations in \mathcal{Z} can, for example, be found in [Knuth 1981, Kannan and Bachem 1979, Iliopoulos 1989a, Iliopoulos 1989b].

¹⁵Every equation in the unification problem yields one linear equation.

Unification with constants

For *ACU*-unification with constants, there are two different ways of extending the approach for elementary unification to the case of unification with constants. The approach originally proposed by Stickel [1975] and [1981] first solves an elementary *ACU*-unification problem, which is obtained by treating free constants as variables, and then modifies the solutions of the elementary problem to obtain solutions of the problem with constants. The other approach, due to Livesey and Siekmann [1975] and described in more detail in [Herold and Siekmann 1987], handles free constants with the help of inhomogeneous linear equations. In the following, we restrict our attention to this second method.

As an example, we slightly modify the *ACU*-unification problem from above. Let $\Gamma'_{ACU} := \{f(x, f(x, y)) = ?_{ACU} f(a, f(z, f(z, z)))\}$, where a is a (free) constant. Of course, the numbers of occurrences $|x\tau|_{z'}$, $|y\tau|_{z'}$, $|z\tau|_{z'}$ of a variable z' introduced by an *ACU*-unifier of this problem must still solve the (homogeneous) linear equation $2x + y = 3z$. For the free constant a , however, one must also take into account that a already occurs once on the right-hand side. Thus, the numbers $|x\tau|_a$, $|y\tau|_a$, $|z\tau|_a$ must solve the following *inhomogeneous* equation:

$$2x + y = 3z + 1.$$

The minimal (non-trivial) nonnegative integer solutions of this equation are $(0, 1, 0)$ and $(2, 0, 1)$. Every nonnegative integer solution of the equation can be obtained as the sum of one of the minimal solution and a solution of the corresponding homogeneous equation $2x + y = 3z$. Consequently, each of the minimal solutions of the inhomogeneous equation together with the set of all minimal solutions of the homogeneous equation gives rise to one element of the minimal complete set of *ACU*-unifiers of the problem:

$$\begin{aligned} & \{ \{x \mapsto f(z_1, f(z_3, f(z_3, z_3))), y \mapsto f(a, f(z_1, f(z_2, f(z_2, z_2)))), \\ & \quad z \mapsto f(z_1, f(z_2, f(z_3, z_3)))\}, \\ & \{x \mapsto f(a, f(a, f(z_1, f(z_3, f(z_3, z_3))))) , y \mapsto f(z_1, f(z_2, f(z_2, z_2))), \\ & \quad z \mapsto f(a, f(z_1, f(z_2, f(z_3, z_3))))\} \}. \end{aligned}$$

In the general case, one must solve one inhomogeneous equation for each free constant occurring in the unification problem. The unifiers in the minimal complete set then correspond to all possible combinations of the minimal solutions of these inhomogeneous equations. For example, if the unification problem contains the free constants a, b, c , and if the sets of minimal solutions of the inhomogeneous equations induced by a, b , and c , respectively, have cardinality 2, 3, and 5, then the minimal complete set is of cardinality $2 \cdot 3 \cdot 5 = 30$.

Unification with constants modulo the theories *ACUI* and *AG* can be treated accordingly. In both cases, one works in the semiring corresponding to the theory, and first determines a generating set of solutions for the system of homogeneous equations corresponding to the unification problem. Then, one considers the systems of

inhomogeneous equations induced by the free constants, and for each system determines finitely many solutions such that all solutions of this system of inhomogeneous equations can be represented as the sum of one of these particular solutions and a solution of the homogeneous equation. From these sets of solutions, the minimal complete set of unifiers can be computed, as illustrated in the above example.

For AG , the fact that the corresponding semiring is a ring implies that taking one particular solution for each system of inhomogeneous equations is sufficient. Consequently, AG is unitary both for elementary unification and for unification with constants, whereas the other two theories, though unitary for elementary unification, are only finitary for unification with constants.

5.2. The class of commutative/monoidal theories

In order to generalize this semantic approach to a whole class of theories, let us try to determine the relevant common features of the theories ACU , $ACUI$, and AG . Using a rather syntactic point of view, we may observe that all three theories are concerned with an associative-commutative binary function symbol f with a unit e . In addition, the signature of AG contains a unary function symbol i , which behaves like an endomorphism for f and e , i.e., $i(f(x, y)) =_{AG} f(i(x), i(y))$ and $i(e) =_{AG} e$. This observation motivates the following definition of monoidal theories [Nutt 1990]:

5.1. DEFINITION. An equational theory E is called *monoidal* iff it satisfies the following properties:

1. $Sig(E)$ contains a binary function symbol f and a constant symbol e , and all other function symbols in $Sig(E)$ are unary.
2. The symbol f is associative-commutative with unit e , i.e., $f(f(x, y), z) =_E f(x, f(y, z))$, $f(x, y) =_E f(y, x)$, and $f(x, e) =_E x$.
3. Every unary function symbol $h \in Sig(E)$ is an endomorphism for f and e , i.e., $h(f(x, y)) =_E f(h(x), h(y))$ and $h(e) =_E e$.

Obviously, the theories ACU , $ACUI$, and AG are monoidal. Other examples of monoidal theories are the theories $E_{h,g} \cup E_{h,e} \cup ACU_g$, $E_{h,g} \cup E_{h,e} \cup ACUI_g$, and $E_{h,g} \cup E_{h,e} \cup AG_g$ introduced in subsection 3.4. The theory of Boolean rings and the theory of commutative rings are not monoidal since their signatures contain *two* binary function symbols.

A drawback of the above definition of monoidal theories is that the signature and the axioms defining a theory play an important rôle. In fact, the theory of Abelian groups allows for many different axiomatizations, some of which do not satisfy the definition of a monoidal theory. For example, let g be a binary function symbol and e be a constant symbol. The theory

$$AG' := \{g(x, x) \approx e, g(x, e) \approx e, g(g(x, g(e, y)), g(e, z)) \approx g(g(z, g(e, y)), g(e, x))\}$$

is not monoidal since g is neither associative nor commutative modulo AG' . Nevertheless, any model of AG' is an Abelian group, where the group operations f and i are defined as $f(x, y) := g(x, g(e, y))$ and $i(x) := g(e, x)$.

In order to capture theories like AG' as well, one must take a more semantic point of view. A common feature of the free algebras defined by ACU , $ACUI$, and AG is that the finitely generated free algebras are *direct powers* of the free algebras in one generator. For example, it is well known that the free Abelian group in one generator is just the additive group of the integers, and that the free Abelian group in n generators is the n -fold direct product of this group. As shown in [Baader 1989b], this common feature can nicely be generalized in the categorical setting introduced in subsection 3.3.3:

5.2. DEFINITION. Let E be an equational theory and $\mathcal{F} := \text{Sig}(E)$. Then E is a *commutative* theory iff $C_{\mathcal{F}}(E)$ is a semi-additive category,¹⁶ i.e.,

1. $C_{\mathcal{F}}(E)$ has a zero object.
2. For every pair of objects in $C_{\mathcal{F}}(E)$, their coproduct is also their product.

In algebraic terms, the first condition means that the initial algebra in $V(E)$, i.e., $\mathcal{T}(\mathcal{F}, \emptyset)/_{=E}$, is of cardinality 1. Since the coproduct of $\mathcal{T}(\mathcal{F}, \mathcal{X})/_{=E}$ and $\mathcal{T}(\mathcal{F}, \mathcal{Y})/_{=E}$ is simply $\mathcal{T}(\mathcal{F}, \mathcal{X} \uplus \mathcal{Y})/_{=E}$ (where \uplus denotes disjoint union), the second condition means that the free algebra $\mathcal{T}(\mathcal{F}, \mathcal{X} \uplus \mathcal{Y})/_{=E}$ is isomorphic to the direct product $\mathcal{T}(\mathcal{F}, \mathcal{X})/_{=E} \times \mathcal{T}(\mathcal{F}, \mathcal{Y})/_{=E}$. In particular, this implies that the finitely generated E -free algebras are direct powers of the E -free algebra in one generator.

The theory of Abelian groups satisfies these properties (and thus is commutative). The theory of Boolean rings and the theory of commutative rings are not commutative in the sense of the above definition since the initial algebras contain two elements (the constants 0 and 1).

In order to obtain a more algebraic definition of commutative theories, which also makes clear that all monoidal theories are commutative, we need two more notions from universal algebra. A constant symbol $e \in \mathcal{F}$ is called *idempotent* in E iff $f(e, \dots, e) =_E e$ holds for all $f \in \mathcal{F}$. Any term $t(x_1, \dots, x_n)$ over the signature \mathcal{F} defines an n -ary *implicit operation* o_t in $V(E)$: for an algebra $\mathcal{A} \in V(E)$, the result of applying o_t to elements a_1, \dots, a_n of the carrier of \mathcal{A} is obtained by evaluating $t(a_1, \dots, a_n)$ in \mathcal{A} . For example, the terms $g(x, g(e, y))$ and $g(e, x)$ define a binary and a unary implicit operation, which together with the constant e satisfy the axioms of Abelian groups in all models of AG' , i.e., all algebras in $V(AG')$.

5.3. PROPOSITION. Let E be an equational theory and $\mathcal{F} := \text{Sig}(E)$. Then E is a *commutative theory* iff

1. The signature \mathcal{F} contains a constant e that is idempotent in E .
2. There is a binary implicit operation $*$ in $V(E)$ such that
 - (a) The constant e is a unit for $*$ in all algebras in $V(E)$.
 - (b) For any n -ary function symbol $h \in \mathcal{F}$, the identity $h(x_1 * y_1, \dots, x_n * y_n) \approx h(x_1, \dots, x_n) * h(y_1, \dots, y_n)$ holds in all algebras in $V(E)$.

¹⁶See, e.g., [Herrlich and Strecker 1973, Baader 1989b] for a more precise definition of and more information on semi-additive categories.

Although it is not explicitly required by the proposition, the implicit operation $*$ turns out to be associative and commutative. Using this proposition, it is easy to show that the theory AG' is indeed commutative: the implicit operation $*$ is defined by the term $g(x, g(e, y))$.

Another easy consequence of the proposition is that every monoidal theory is commutative: just take the explicit associative-commutative binary operation f in the definition of monoidal theories as the implicit operation $*$. The theory AG' is an example of a commutative theory that is not monoidal. However, it can be shown [Baader and Nutt 1996] that every commutative theory can be turned into an “equivalent” monoidal theory with the help of a signature transformation. For this reason, one can in principle use both notions synonymously.

5.3. The corresponding semiring

Let E be a commutative theory with $\text{Sig}(E) = \mathcal{F}$. The semiring S_E corresponding to E is obtained by considering the E -free algebra in one generator, say x , and then taking the set of all endomorphisms of this algebra. Each such endomorphism is uniquely determined by the image of the generator x . The multiplication operation “ \cdot ” in S_E is just composition of morphisms, and the addition operation “ $+$ ” is obtained by argument-wise application of the implicit operation $*$ of the commutative theory E : $(\sigma + \tau)(x) := \sigma(x) * \tau(x)$.

As an example, we consider the commutative theory $ACUI$, where the explicit operation f serves as the implicit operation $*$. Since the $ACUI$ -free algebra generated by x consists of two equivalence classes, with representatives x and e , respectively, there are two possible endomorphisms: 0 , which is defined by $x \mapsto e$, and 1 , which is defined by $x \mapsto x$. It is easy to see that the operation “ $+$ ” in S_{ACUI} behaves like disjunction and “ \cdot ” like conjunction on the truth values 0 and 1 . For example, $(0 \cdot 1)(x) = 1(0(x)) = 1(e) = e = 0(x)$ and $(0 + 1)(x) = f(0(x), 1(x)) = f(e, x) =_{ACUI} x = 1(x)$. Consequently, S_{ACUI} is the two-element Boolean semiring BS .

A well-known result for semi-additive categories [Herrlich and Strecker 1973] says that morphisms σ in the semi-additive category $C_{\mathcal{F}}(E)$ can be represented as matrices M_{σ} over S_E such that composition of morphisms corresponds to matrix multiplication, i.e., $M_{\sigma\tau} = M_{\sigma} \cdot M_{\tau}$. For example, the morphism $\sigma: \mathcal{T}(\mathcal{F}, \{x_1, x_2\}) /_{=_{ACUI}} \rightarrow \mathcal{T}(\mathcal{F}, \{y_1, y_2\}) /_{=_{ACUI}}$ defined by $\sigma(x_1) := f(y_1, y_2)$, $\sigma(x_2) := y_2$ corresponds to the matrix

$$M_{\sigma} = \begin{pmatrix} \{x_1 \mapsto y_1\} & \{x_1 \mapsto y_2\} \\ \{x_2 \mapsto e\} & \{x_2 \mapsto y_2\} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

The second equality depends on the fact that all E -free algebras in one generator are isomorphic, and thus a morphism $\sigma_{ij}: \mathcal{T}(\mathcal{F}, \{x_i\}) /_{=E} \rightarrow \mathcal{T}(\mathcal{F}, \{y_j\}) /_{=E}$ can be seen as an endomorphism of $\mathcal{T}(\mathcal{F}\{x\}) /_{=E}$, i.e., an element of S_E .

5.4. Results on unification in commutative theories

Let E be a commutative theory with $\text{Sig}(E) = \mathcal{F}$. In subsection 3.3.3 we have seen that any E -unification problem over \mathcal{F} corresponds to a parallel pair $\sigma, \tau: \mathcal{T}(\mathcal{F}, \mathcal{I}) /_{=E} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X}) /_{=E}$ of morphisms in $C_{\mathcal{F}}(E)$, and that an E -unifier corresponds to a morphism δ with domain $\mathcal{T}(\mathcal{F}, \mathcal{X}) /_{=E}$ such that $\sigma\delta = \tau\delta$ holds in $C_{\mathcal{F}}(E)$.

If we translate the morphisms into matrices over S_E , this means that an E -unifier of the parallel pair $\langle \sigma, \tau \rangle$ corresponds to a matrix M over S_E such that $M_{\sigma} \cdot M = M_{\tau} \cdot M$. This correspondence is used in [Nutt 1990, Baader 1993] to characterize the unification types of commutative theories by algebraic properties of the corresponding semirings. The rows of the matrix M are n -tuples of elements of S_E , written as row vectors. We will denote the set of all such n -dimensional row vectors over S_E by S_E^n .

5.4. THEOREM. *A commutative theory E is unitary w.r.t. elementary unification iff the corresponding semiring S_E satisfies the following condition: for all $m, n \geq 1$ and all $m \times n$ -matrices M_1, M_2 over S_E the set*

$$U(M_1, M_2) := \{v \in S_E^n \mid M_1 \cdot v = M_2 \cdot v\}$$

is finitely generated, i.e., there exist $k \geq 0$ and $v_1, \dots, v_k \in S_E^n$ such that $U(M_1, M_2) = \{v_1 \cdot s_1 + \dots + v_k \cdot s_k \mid s_1, \dots, s_k \in S_E\}$.

If $\{v_1, \dots, v_k\}$ is such a finite generating set for $U(M_{\sigma}, M_{\tau})$, then the matrix whose columns are the vectors v_1, \dots, v_k corresponds to the most general E -unifier of $\langle \sigma, \tau \rangle$.

Unification with constants can also be reformulated as a problem in $C_{\mathcal{F}}(E)$ for $\mathcal{F} = \text{Sig}(E)$. To this end we view constants as special variables that must always be substituted for themselves. Let \mathcal{C} be a finite set of free constants. We say that a morphism $\sigma: \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{C}) /_{=E} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{Y} \cup \mathcal{C}) /_{=E}$ respects the constants in \mathcal{C} iff $c\sigma = c$ for all $c \in \mathcal{C}$. In this case, the matrix M_{σ} has a special form:

$$M_{\sigma} = \begin{pmatrix} M_{\sigma}^h & M_{\sigma}^i \\ 0 & U \end{pmatrix},$$

where M_{σ}^h is an $|\mathcal{X}| \times |\mathcal{Y}|$ -matrix, M_{σ}^i is an $|\mathcal{X}| \times |\mathcal{C}|$ -matrix, 0 is the $|\mathcal{C}| \times |\mathcal{Y}|$ -matrix with all entries 0 , and U is the $|\mathcal{C}| \times |\mathcal{C}|$ -unit matrix. The 0 -submatrix is due to the fact that σ does not substitute terms with variables for constants, and the unit matrix expresses that σ maps any constant to itself.

An E -unification problem with constants from a finite set \mathcal{C} corresponds to a parallel pair $\langle \sigma, \tau \rangle$ of morphisms respecting the constants in \mathcal{C} , and each E -unifier δ of this pair also corresponds to a morphism respecting \mathcal{C} . For the components of the corresponding matrices, the fact that δ is a unifier of $\langle \sigma, \tau \rangle$, i.e., that $M_{\sigma} \cdot M_{\delta} = M_{\tau} \cdot M_{\delta}$, leads to the following equations:

$$M_{\sigma}^h M_{\delta}^h = M_{\tau}^h M_{\delta}^h,$$

$$M_{\sigma}^h M_{\delta}^i + M_{\sigma}^i = M_{\tau}^h M_{\delta}^i + M_{\tau}^i.$$

The first equation is a system of homogeneous equations in S_E , whereas the second is a system of inhomogeneous equations.

From these observations one can derive the following characterization of the type “at most finitary” for unification with constants in commutative theories:¹⁷

5.5. THEOREM. *Let E be a commutative theory that is unitary w.r.t. elementary unification. Then E is at most finitary w.r.t. unification with constants iff the corresponding semiring S_E satisfies the following condition: for all $m, n \geq 1$, all $m \times n$ -matrices M_1, M_2 over S_E , and all $u_1, u_2 \in S_E^m$, there exist finitely many $v_1, \dots, v_k \in S_E^n$ such that*

$$\{w \in S_E^n \mid M_1 \cdot w + u_1 = M_2 \cdot w + u_2\} = \{v_i + v \mid 1 \leq i \leq k, v \in U(M_1, M_2)\}.$$

This condition means that finitely many particular solutions of the system of inhomogeneous equations, $M_1 \cdot x + u_1 = M_2 \cdot x + u_2$, together with the solutions $U(M_1, M_2)$ of the corresponding system of homogeneous equations, $M_1 \cdot x = M_2 \cdot x$, generate all solutions of the system of inhomogeneous equations. The assumption that E is unitary w.r.t. elementary unification implies that $U(M_1, M_2)$ is finitely generated. The complete set of E -unifiers can now be built from the generating set of $U(M_1, M_2)$ and the finitely many particular solutions of the systems of inhomogeneous equations corresponding to the free constants as illustrated in subsection 5.1.

We close this section by mentioning some additional results on unification in commutative theories. Let E be a commutative theory.

1. For elementary unification, E is either unitary or of type zero.
2. If S_E is finite, then E is unitary for elementary unification and at most finitary for unification with constants.
3. If S_E is a ring and E is unitary for elementary unification, then E is also unitary for unification with constants.
4. If E is at most finitary for unification with constants, then E is also at most finitary for unification with linear constant restrictions, and thus also for general unification.

Proofs of these and other interesting results on unification in commutative/monoidal theories can be found in [Baader 1989b, Nutt 1990, Baader 1993, Baader and Nutt 1996].

Compared to syntactic approaches to unification, the semantic approach introduced here has the disadvantage that it cannot treat general unification problems directly. In fact, for a commutative theory E , we have considered the category $C_{\mathcal{F}}(E)$ for $\mathcal{F} = \text{Sig}(E)$, and have used the fact that this category is semi-additive. For an extended signature $\mathcal{F}_1 \supset \mathcal{F}$, the category $C_{\mathcal{F}_1}(E)$ would no longer be semi-additive, and thus the presented approach to unification in commutative theories cannot be applied directly. For unification with constants, we have shown that one can still work within the category $C_{\mathcal{F}}(E)$ by considering special morphisms. For

¹⁷Recall that “at most finitary” means unitary or finitary.

arbitrary free function symbols such an approach does not appear to be possible. The general methods for combining unification algorithms described in the next section can, however, overcome this problem (see result 4. from above).

6. Combination of unification algorithms

In applications of equational unification in automated deduction, one is often faced with the problem of unifying terms containing several function symbols whose properties are defined by equational theories. For example, associative-commutative function symbols often come in pairs (e.g., the addition operation $+$ and the multiplication operation $*$ of rings). However, a given *AC*- or *ACU*-unification algorithm can only treat terms containing one of these two symbols, but not both. In program verification one may encounter data structures such as sets and lists, and their combination (e.g., sets of lists). Since union of sets (\cup) is associative, commutative, and idempotent, and the append operation for lists (*app*) is associative, unification of terms containing both *ACI*- and *A*-symbols is of interest in this setting. Thus, the question arises whether we can use the known *ACI* $_{\cup}$ - and A_{app} -unification algorithms for unifying terms containing both \cup and *app* modulo $ACI_{\cup} \cup A_{app}$. This is an instance of the following *combination problem for unification algorithms*:

Assume that E_1, \dots, E_n are equational theories over pairwise disjoint signatures. How can algorithms for unification modulo E_i ($i = 1, \dots, n$) be combined to an algorithm for unification modulo $E_1 \cup \dots \cup E_n$?

To be more precise, there are two variants of this problem: one can either try to combine algorithms computing complete sets of unifiers or decision procedures. It should also be noted that without the disjointness condition there cannot exist a general combination method.¹⁸ For example, as mentioned in section 3.4, $D_{f,g}^l$ -unification and $D_{f,g}^r$ -unification are unitary, whereas unification modulo their union $D_{f,g}$ is infinitary, which shows that algorithms computing finite complete sets of unifiers cannot be combined in the non-disjoint case. Section 3.4 also yields a negative example for the combination of decision procedures: $D_{f,g}$ -unification and A_g -unification are decidable, whereas unification modulo their union is undecidable.

The formulation of the combination problem given above is still not quite precise since it does not specify which kind of E_i -unification problems (elementary, with constants, or general) the component algorithms must be able to handle. As we shall see below, algorithms for unification with constants are not quite sufficient: the combination method requires algorithms for unification with linear constant restrictions for the component theories E_i . In particular, algorithms for general E -unification can be obtained from algorithms for E -unification with lcr by combining them with an algorithm for syntactic unification (which treats the free function symbols).

¹⁸There are some approaches that try to weaken the disjointness assumption, but the theories to be combined must satisfy rather strong conditions [Ringeissen 1992, Domenjoud, Klay and Ringeissen 1994].

The research on the combination problem was triggered by the search for a unification algorithm that can deal with terms containing several associative-commutative function symbols and free symbols [Stickel 1975, Stickel 1981, Fages 1984, Fages 1987, Herold and Siekmann 1987]. It turned out that the methods used in this particular instance of the combination problem can easily be generalized to other equational theories, provided that they satisfy certain restrictions (such as collapse-freeness or regularity¹⁹) on the syntactic form of their defining identities, which make sure that the theories behave similarly to associativity-commutativity and syntactic equality [Kirchner 1985, Tidén 1986, Herold 1986, Yelick 1987, Boudet et al. 1989].

The problem of combining algorithms computing complete sets of unifiers was solved in a very general form by Schmidt-Schauß [1989b]. His approach imposes no restriction on the syntactic form of the identities. The only requirements on the component theories E_i are of an algorithmic nature: both E_i -unification problems with constants and so-called “constant elimination problems” (see [Schmidt-Schauß 1989b] for a definition) must be finitary solvable modulo E_i . Boudet [1993] describes a more efficient combination algorithm, which depends on the same requirements as the one by Schmidt-Schauß.

In the following, we will describe the combination method introduced in [Baader and Schulz 1992, Baader and Schulz 1996] in more detail, since it can be used both for combining algorithms computing complete sets of unifiers and for combining decision procedures. Instead of splitting the algorithmic problem to be solved for the component theories E_i into two parts (unification with constants and constant elimination), this method requires algorithms (decision procedures) for E_i -unification with lcr. In this setting, Schmidt-Schauß’s condition that constant elimination problems must be finitary solvable modulo E_i can be seen as just one way of ensuring that E_i -unification with lcr is at most finitary provided that E_i -unification with constants is at most finitary.

6.1. A general combination method

Before describing the combination method of Baader and Schulz [1992] and [1996] formally, we illustrate the underlying ideas by a simple example. Let g be a unary and f be a binary function symbol. We consider the theories A_f and $F_g := \{g(x) \approx g(x)\}$,²⁰ and the (elementary) unification problem

$$\Gamma_0 := \{g(f(y, y)) \stackrel{?}{=} g(x), g(x) \stackrel{?}{=} g(y), x \stackrel{?}{=} f(y, y)\}$$

modulo their union $E := A_f \cup F_g$. In a first step, we transform Γ_0 into an equivalent unification problem in decomposed form, i.e., into a union of an (elementary) A_f -

¹⁹A theory E is called collapse-free if it does not contain an identity of the form $x = t$ where x is a variable and t is a non-variable term, and it is called regular if the left- and right-hand sides of the identities contain the same variables.

²⁰Obviously, \approx_{F_g} is just syntactic equality. The “dummy” axiom $g(x) \approx g(x)$ makes sure that g belongs to $\text{Sig}(F_g)$.

unification problem and an (elementary) F_g -unification problem:

$$\Gamma := \{z \stackrel{?}{A_f} f(y, y), x \stackrel{?}{A_f} f(y, y)\} \cup \{g(z) \stackrel{?}{F_g} g(x), g(x) \stackrel{?}{F_g} g(y)\}.$$

This has been achieved by replacing “alien” subterms (in the example, just the term $f(y, y)$ occurring on the left-hand side of the first equation) by new variables and introducing appropriate new equations (see [Baader and Schulz 1996] for a formal definition of this decomposition step).

Unfortunately, it is not sufficient simply to test the “pure” unification problems obtained this way for solvability. The problem is that these unification problems still share variables, and the single solutions may instantiate these variables with incompatible terms. For example, $\sigma_1 := \{x \mapsto f(y, y), z \mapsto f(y, y)\}$ solves the A_f -subproblem, and $\sigma_2 := \{x \mapsto g(x), y \mapsto g(x), z \mapsto g(x)\}$ is a solution of the F_g -subproblem, but these solutions replace both x and z by different (even non-unifiable) terms. In order to avoid such incompatible assignments, we choose a theory label for each variable: in the subproblem corresponding to this theory, the variable may be instantiated, whereas in the other subproblem the variable must be treated as a constant. For example, if we assign

$$L(x) := L(z) := A_f \text{ and } L(y) := F_g,$$

then y must be treated as a constant in the A_f -subproblem, whereas x and z must be treated as constants in the F_g -subproblem.

This avoids incompatible instantiations of shared variables, but also leads to a new problem: in the example, the equation $g(z) \stackrel{?}{F_g} g(x)$ is no longer solvable since both z and x must be treated as (different) constants. This problem can be overcome by choosing an appropriate variable identification. In the example, x must be identified with z , which can be achieved by replacing every occurrence of z by x :

$$\Gamma' := \{x \stackrel{?}{A_f} f(y, y)\} \cup \{g(x) \stackrel{?}{F_g} g(x), g(x) \stackrel{?}{F_g} g(y)\}.$$

Unfortunately, the solutions $\sigma'_1 := \{x \mapsto f(y, y)\}$ and $\sigma'_2 := \{y \mapsto x\}$ of the pure subproblems still cannot be combined to a solution of their union, since there is a cyclic dependency between the two substitutions: x is replaced by a term containing y , and y is replaced by a term containing x . Such cyclic dependencies between solutions of the pure subproblems can finally be avoided by choosing a linear ordering on the shared variables of the unification problem, which induces linear constant restrictions for the subproblems.

These ideas can be formalized as follows. Let E_1, \dots, E_n be non-trivial equational theories over disjoint signatures. An $(E_1 \cup \dots \cup E_n)$ -unification problem Γ is in *decomposed form* iff $\Gamma = \Gamma_1 \cup \dots \cup \Gamma_n$ where each Γ_i is an elementary E_i -unification problem. As illustrated in the example, it is easy to transform a given elementary $(E_1 \cup \dots \cup E_n)$ -unification problem into an equivalent problem in decomposed form (see [Baader and Schulz 1996] for details). Thus, we may without loss of generality assume that all our $(E_1 \cup \dots \cup E_n)$ -unification problems are in decomposed form

$\Gamma = \Gamma_1 \cup \dots \cup \Gamma_n$. A variable occurring in Γ is called a *shared variable* iff it occurs in at least two of the pure subproblems Γ_i .

Let \mathcal{X} be the set of shared variables of $\Gamma = \Gamma_1 \cup \dots \cup \Gamma_n$. A *variable identification* can be represented by a partition $\Pi = \{P_1, \dots, P_k\}$ of \mathcal{X} . For each of the classes P_i , let $x_i \in P_i$ be a representative of this class, and let $\mathcal{X}_\Pi := \{x_1, \dots, x_k\}$ be the set of these representatives. The substitution that replaces, for all $i = 1, \dots, k$, each element of P_i by its representative x_i is denoted by σ_Π . We denote the result of applying σ_Π to each term in Γ_i by $\Gamma_i\sigma_\Pi$. For a given partition Π of the shared variables of Γ , let $L : \mathcal{X}_\Pi \rightarrow \{1, \dots, n\}$ be a *labelling function*, which assigns a theory label to each variable in \mathcal{X}_Π , and let $<$ be a *linear ordering* on \mathcal{X}_Π . Using L and $<$, each of the elementary E_i -unification problems $\Gamma_i\sigma_\Pi$ can be turned into an E_i -unification problem with linear constant restrictions $\langle \Gamma_i\sigma_\Pi, L, < \rangle$: the variables $x \in \mathcal{X}_\Pi$ with label $L(x) \neq i$ are treated as (free) constants in $\langle \Gamma_i\sigma_\Pi, L, < \rangle$, whereas the other variables are still treated as variables, and the linear constant restrictions are induced by $<$.²¹

6.1. PROPOSITION. *Let $\Gamma := \Gamma_1 \cup \dots \cup \Gamma_n$ be an $(E_1 \cup \dots \cup E_n)$ -unification problem in decomposed form. Then the following statements are equivalent:*

1. Γ is solvable, i.e., there exists an $(E_1 \cup \dots \cup E_n)$ -unifier of Γ .
2. There exists a partition Π , a labelling function $L : \mathcal{X}_\Pi \rightarrow \{1, \dots, n\}$, and a linear ordering $<$ on \mathcal{X}_Π such that, for all $i = 1, \dots, n$, the E_i -unification problem with linear constant restrictions $\langle \Gamma_i\sigma_\Pi, L, < \rangle$ is solvable.

Assume that solvability of E_i -unification problems with lcr is decidable for $i = 1, \dots, n$. For a given elementary $(E_1 \cup \dots \cup E_n)$ -unification problem Γ_0 one can compute an equivalent problem in decomposed form Γ in polynomial time. For Γ , there exist only finitely many different triples $(\Pi, L, <)$, which means that it is possible to compute all possible such triples, and then test the obtained E_i -unification problems with lcr for solvability. Thus, proposition 6.1 implies that solvability of elementary $(E_1 \cup \dots \cup E_n)$ -unification problems is decidable. To be more precise, instead of deterministically computing all possible triples $(\Pi, L, <)$, one can also employ a non-deterministic algorithm that “guesses the right tuple” in polynomial time.

6.2. THEOREM. *Let E_1, \dots, E_n be non-trivial equational theories over disjoint signatures. If solvability of E_i -unification problems with linear constant restrictions is decidable (in NP) for $i = 1, \dots, n$, then solvability of elementary $(E_1 \cup \dots \cup E_n)$ -unification problems is decidable (in NP).*

In general, it is not possible to avoid the non-determinism inherent in this combination method [Schulz 1997]. For example, the decision problem is polynomial for ACUI-unification with lcr, but NP-complete for general ACUI-unification [Baader

²¹Non-shared variables are assumed to be larger than all shared variables, i.e., there are no restrictions for the images of these variables.

and Schulz 1993b, Kapur and Narendran 1992a]. This shows that the combination of an algorithm for syntactic unification with a decision procedure for *ACUI*-unification with lcr cannot be achieved with the help of a polynomial combination method. For regular and collapse-free theories for which, in addition, it is possible to compute most general unifiers in polynomial time, one can, however, design a (deterministic) polynomial combination procedure [Schulz 1999].

The naive combination algorithm obtained by a direct application of proposition 6.1 is highly non-deterministic, and thus does not lead to satisfactory results in practice. Optimizations of the combination algorithm (which avoid this unsatisfactory behavior in many cases) are described in [Kepser and Richts 1999].

Proposition 6.1 can also be used to obtain a method for combining unification algorithms, i.e., algorithms computing finite complete sets of unifiers. In fact, as we shall see below, given solutions σ_i of the E_i -unification problems with lcr induced by the triple $(\Pi, L, <)$ can effectively be combined into a solution $\sigma_1 \odot \dots \odot \sigma_n$ of the original $(E_1 \cup \dots \cup E_n)$ -unification problem. For a given $(E_1 \cup \dots \cup E_n)$ -unification problem Γ in decomposed form, let T_1, \dots, T_k be all the triples consisting of a partition Π , a labelling function L , and a linear ordering $<$ on \mathcal{X}_Π , and let $C_{i,j}$ be a complete set of E_i -unifiers of the E_i -unification problem with lcr induced by T_j . Then the set

$$\bigcup_{j=1}^k \{ \sigma_1 \odot \dots \odot \sigma_n \mid \sigma_i \in C_{i,j} \}$$

is a complete set of $(E_1 \cup \dots \cup E_n)$ -unifiers of Γ (see [Baader and Schulz 1996] for a proof).

6.3. THEOREM. *Let E_1, \dots, E_n be non-trivial equational theories over disjoint signatures that are at most finitary for E_i -unification with linear constant restrictions. Then $E_1 \cup \dots \cup E_n$ is at most finitary for elementary unification.*

Although the combination results (as formulated in theorem 6.2 and theorem 6.3) only apply to elementary unification in the combined theory, they can easily be extended to general unification. In fact, it is easy to see that syntactic unification with lcr is decidable and unitary: just compute the mgu of the unification problem without lcr, and then test whether it satisfies the constant restrictions. Thus, one can simply take as one of the E_i 's a "free" theory F such that $\text{Sig}(F)$ contains all the free function symbols occurring in the general unification problem and $=_F$ is the syntactic equality on $\text{Sig}(F)$ -terms.

6.2. Proving correctness of the combination method

In order to show *soundness of the combination method* (i.e., (2) \rightarrow (1) of proposition 6.1), it is sufficient to show that given solutions σ_i of the E_i -unification problems with lcr induced by the triple $(\Pi, L, <)$ can indeed be combined into a

solution $\sigma_1 \odot \cdots \odot \sigma_n$ of the original $(E_1 \cup \cdots \cup E_n)$ -unification problem in decomposed form $\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_n$. First, we combine $\sigma_1, \dots, \sigma_n$ into a solution σ of $\Gamma\sigma_\Pi = \Gamma_1\sigma_\Pi \cup \cdots \cup \Gamma_n\sigma_\Pi$. Obviously, this implies that $\sigma_\Pi\sigma$ is a solution of Γ .

Without loss of generality, we may assume that the substitution σ_i maps all variables with label i to terms containing only variables with label $j \neq i$ (which are treated as free constants in $\Gamma_i\sigma_\Pi$) or new variables, i.e., variables not occurring in Γ . The combined solution σ of $\Gamma\sigma_\Pi$ is defined along the linear ordering $<$.

Let x be the least variable with respect to $<$, and let i be its label. Since the solution σ_i of $\Gamma_i\sigma_\Pi$ satisfies the constant restrictions induced by $<$, the term $x\sigma_i$ does not contain any variables with index $j \neq i$. Thus we can simply define $x\sigma := x\sigma_i$.

Now let x be an arbitrary variable with label i , and let y_1, \dots, y_m be the variables with labels different from i occurring in $x\sigma_i$. Since σ_i satisfies the constant restrictions induced by $<$, the variables y_1, \dots, y_m (which are treated as free constants in $\Gamma_i\sigma_\Pi$) must be smaller than x . This means that $y_1\sigma, \dots, y_m\sigma$ are already defined. The term $x\sigma$ is now obtained from $x\sigma_i$ by replacing each y_k by $y_k\sigma$ ($k = 1, \dots, m$).

It is easy to see that the substitution σ obtained this way satisfies $\sigma = \sigma_i\sigma$ ($i = 1, \dots, n$), i.e., σ is an instance of all the substitutions σ_i . Since σ_i is an E_i -unifier of $\Gamma_i\sigma_\Pi$, this implies that σ is also an E_i -unifier of $\Gamma_i\sigma_\Pi$, and thus an E -unifier of $\Gamma_i\sigma_\Pi$. Consequently, σ is an E -unifier of $\Gamma\sigma_\Pi = \Gamma_1\sigma_\Pi \cup \cdots \cup \Gamma_n\sigma_\Pi$.

Proving *completeness of the combination method* (i.e., (1) \rightarrow (2) of proposition 6.1) turns out to be a bit more complex. In the following, we only give a sketch of the proof. Assume that σ is a solution of the $(E_1 \cup \cdots \cup E_n)$ -unification problem in decomposed form $\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_n$. This solution can be used to define the correct triple $(\Pi, L, <)$:

1. Two shared variables x, y belong to the same class of Π iff $x\sigma =_E y\sigma$.
2. If $x\sigma$ is not a variables, then $L(x) = i$ iff the top symbol of $x\sigma$ belongs to $Sig(E_i)$. Otherwise, $L(x) := 1$ (this is an arbitrary decision).
3. $<$ is an arbitrary linear extension of the strict partial ordering \prec defined by $x \prec y$ iff $x\sigma$ is a strict subterm of $y\sigma$.

It is easy to see that σ is also a solution of $\Gamma\sigma_\Pi = \Gamma_1\sigma_\Pi \cup \cdots \cup \Gamma_n\sigma_\Pi$. For each i , the substitution σ (which is a substitution of the combined signature $Sig(E_1) \cup \cdots \cup Sig(E_n)$) can be turned into a $Sig(E_i)$ -substitution σ_i by replacing alien subterms in $x\sigma$ (i.e., subterms starting with a symbol not belonging to $Sig(E_i)$) by new variables in such a way that $=_E$ -equivalent subterms are replaced by the same variable. Unfortunately, for an arbitrary E -unifier σ of Γ , the substitution σ_i obtained this way need not be a solution of the E_i -unification problem with lcr $(\Gamma_i\sigma_\Pi, L, <)$. For this to be true, σ must be normalized in a certain way. One possibility to obtain an appropriate notion of a normalized substitution is to apply unifying completion to the equational theory $E_1 \cup \cdots \cup E_n$, and normalize w.r.t. the ordered rewrite system R obtained this way (see [Baader and Schulz 1996] for details). Since R may be infinite, it is not necessarily possible to compute the normal form of a given term, but this is irrelevant for the proof of completeness. Another possibility (which has the advantage that normalization is effective) is to

compute a so-called “layer-reduced” form [Schmidt-Schauß 1989b, Kirchner and Ringeissen 1994]. In principles, this normal form is obtained by applying collapse-equations as much as possible.

A different way of proving soundness and completeness of the combination method described above was introduced in [Baader and Schulz 1995a]: it depends on a representation of the free algebra in $V(E_1 \cup \dots \cup E_n)$ over countably many generators as the so-called free amalgamated product of the free algebras in $V(E_i)$ in countably many generators. This approach can also deal with the combination of constraint solvers in free structures (where the signature may also contain predicate symbols), and it has been generalized to structures that are not necessarily free [Baader and Schulz 1995c, Baader and Schulz 1998]. The combination method has also been extended to disunification [Baader and Schulz 1995b, Kepser 1999].

7. Further topics

In this article we have concentrated on unification of first-order terms, and have mentioned only applications in term rewriting and resolution-based theorem proving. However, unification is a broad paradigm with applications in almost every area of automated deduction, and we would like to draw the reader’s attention in particular to the two chapters of this handbook where varieties of unification not covered here are treated: higher-order unification [Dowek 2001] and rigid E -unification [Degtyarev and Voronkov 2001a] (Chapters 16 and 10 of this Handbook). In addition, we briefly mention in this final section a number of important variants of the unification problem that have been studied in the literature.

Matching

Given a pair of terms s, t , the matching problem asks for a substitution σ such that $s\sigma = t$. Again, this syntactic matching problem can be generalized to matching modulo an equational theory E , where one asks for a substitution σ satisfying $s\sigma =_E t$.

If t does not contain variables, then matching and unification are obviously the same problem. In general, one can turn a given matching problem into an “equivalent” unification problem by replacing the variables in t by new free constants. This transformation shows that matching modulo E can be reduced to E -unification *with constants*. Bürckert [1989] has shown that there exists an equational theory for which elementary unification is decidable, but matching and unification with constants is undecidable. Also, if one is interested in complete sets of E -matchers, then one must be careful how to define the instantiation quasi-ordering [Bürckert 1989].

Semiunification

Semiunification is a deceptively simple combination of syntactic matching and syntactic unification on first-order terms.

A *semiunification problem* consists of a set of pairs of terms

$$\{s_1 \leq^? t_1, \dots, s_n \leq^? t_n\}$$

and is called *uniform* if $n = 1$. A substitution σ is a solution (a *semiunifier*) of such a problem iff there exist substitutions ρ_1, \dots, ρ_n such that

$$s_1 \sigma \rho_1 = t_1 \sigma, \dots, s_n \sigma \rho_n = t_n \sigma.$$

This simple definition belies the broad variety of applications of semiunification in term rewriting, type checking for programming languages, proof theory, and computational linguistics; in addition, proving the properties of the problem turned out to be extremely difficult. Although it is easy to show that so-called principal solutions (analogous to *mgus* in syntactic unification) always exist for solvable semiunification problems, the proof that the non-uniform case is undecidable is exceedingly complex; the interested reader is referred to [Kfoury, Tiuryn and Urzyczyn 1993], where a review of the results on the non-uniform case is presented.

The uniform case is decidable, but it took a long time to develop a correct, efficient algorithm. A fast algorithm based on the unification-closure method, as well as a review of the various attempts to provide algorithms for the problem, may be found in [Oliart and Snyder 1998]. This paper shows that the uniform case can be decided in $O(n^2 \alpha(n)^2)$, where n is the size of the two input terms, and α is the functional inverse of Ackermann's function; constructing a principal solution is somewhat more complex.

Disunification

A *disunification problem* is of the form

$$\{s_1 \stackrel{?}{=} t_1, \dots, s_n \stackrel{?}{=} t_n, s_{n+1} \stackrel{?}{\neq} t_{n+1}, \dots, s_{n+m} \stackrel{?}{\neq} t_{n+m}\},$$

where s_1, \dots, t_{n+m} are terms. A solution of such a problem is a substitution σ satisfying $s_i \sigma = t_i \sigma$ ($i = 1, \dots, n$) and $s_{n+j} \sigma \neq t_{n+j} \sigma$ ($j = 1, \dots, m$). Again, this problem can be generalized to disunification modulo an equational theory E .

In contrast to unification, one must distinguish between different types of solvability: for disunification it makes a difference whether solutions are required to be ground substitutions (i.e., substitution introducing only variable-free terms), or whether they may be arbitrary substitutions. Both types of solvability have been considered in the literature [Colmerauer 1984, Kirchner and Lescanne 1987, Bürckert 1988, Comon and Lescanne 1989, Comon 1988, Comon 1991, Buntine and Bürckert 1994, Baader and Schulz 1993a], but ground solvability appears to be more interesting for most applications. It should also be noted that sometimes more general problems than the one introduced above are still called disunification problems (see, e.g., [Comon 1991]).

Sorted unification

In many applications, the domain on which the function symbols operate is not one homogeneous set: it is divided into different subsets, which on the syntactic

level are represented as sorts. Sorted unification generalizes syntactic unification in that the domain of variables is restricted to certain sorts. Unifiers are then required to be well-sorted in the sense that variables can only be replaced by terms of a "compatible" sort. Results for sorted unification strongly depend on the expressiveness of the sort language. An overview on sorted unification can, for example, be found in [Weidenbach 1998]; other important references on the topic are [Walther 1983, Walther 1987, Schmidt-Schauß 1986a, Schmidt-Schauß 1989a, Comon 1989, Meseguer, Goguen and Smolka 1989, Tommasi 1991, Frisch and Cohn 1992, Weidenbach 1996].

Bibliography

- ABDULRAB H. AND PÉCUCHE J.-P. [1989], 'Solving word equations', *J. Symbolic Computation* 8(5), 499–521.
- AUFFRAY Y. AND ENJALBERT P. [1992], 'Modal theorem proving: An equational viewpoint', *J. Logic and Computation* 2(3), 247–295.
- BAADER F. [1986], 'Unification in idempotent semigroups is of type zero', *J. Automated Reasoning* 2(3), 283–286.
- BAADER F. [1989a], Characterizations of unification type zero, in N. Dershowitz, ed., 'Proceedings of the 3rd International Conference on Rewriting Techniques and Applications', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Chapel Hill, North Carolina, pp. 2–14.
- BAADER F. [1989b], 'Unification in commutative theories', *J. Symbolic Computation* 8(5), 479–497.
- BAADER F. [1991], Unification, weak unification, upper bound, lower bound, and generalization problems, in R. V. Book, ed., 'Proceedings of the 4th International Conference on Rewriting Techniques and Applications', Vol. 488 of *Lecture Notes in Computer Science*, Springer-Verlag, Como, Italy, pp. 86–97.
- BAADER F. [1993], 'Unification in commutative theories, Hilbert's basis theorem and Gröbner bases', *J. of the ACM* 40(3), 477–503.
- BAADER F. [1998], 'On the complexity of Boolean unification', *Information Processing Letters* 67(4), 215–220.
- BAADER F. AND BÜTTNER W. [1988], 'Unification in commutative idempotent monoids', *Theoretical Computer Science* 56(1), 345–352.
- BAADER F. AND NARENDRA P. [1998], Unification of concept terms in description logics, in H. Prade, ed., 'Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)', John Wiley & Sons Ltd, Brighton, UK, pp. 331–335.
- BAADER F. AND NUTT W. [1996], 'Combination problems for commutative/monoidal theories: How algebra can help in equational reasoning', *J. Applicable Algebra in Engineering, Communication and Computing* 7(4), 309–337.
- BAADER F. AND SCHULZ K. U. [1992], Unification in the union of disjoint equational theories: Combining decision procedures, in D. Kapur, ed., 'Proceedings of the 11th International Conference on Automated Deduction', Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Saratoga Springs, NY, USA, pp. 50–65.
- BAADER F. AND SCHULZ K. U. [1993a], Combination techniques and decision problems for dis-unification, in C. Kirchner, ed., 'Proceedings of the 5th International Conference on Rewriting Techniques and Applications', *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Montreal, Canada, pp. 301–315.
- BAADER F. AND SCHULZ K. U. [1993b], General A- and AX-unification via optimized combination procedures, in 'Proceedings of the Second International Workshop on Word Equations and

- Related Topics', Vol. 677 of *Lecture Notes in Computer Science*, Springer-Verlag, Rouen, France, pp. 23–42.
- BAADER F. AND SCHULZ K. U. [1995a], Combination of constraint solving techniques: An algebraic point of view, in 'Proceedings of the 6th International Conference on Rewriting Techniques and Applications', Vol. 914 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Kaiserslautern, Germany, pp. 352–366.
- BAADER F. AND SCHULZ K. U. [1995b], 'Combination techniques and decision problems for disunification', *Theoretical Computer Science* **142**, 229–255.
- BAADER F. AND SCHULZ K. U. [1995c], On the combination of symbolic constraints, solution domains, and constraint solvers, in 'Proceedings of the International Conference on Principles and Practice of Constraint Programming, CP95', Vol. 976 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Cassis, France, pp. 380–397.
- BAADER F. AND SCHULZ K. U. [1996], 'Unification in the union of disjoint equational theories: Combining decision procedures', *J. Symbolic Computation* **21**, 211–243.
- BAADER F. AND SCHULZ K. U. [1998], 'Combination of constraint solvers for free and quasi-free structures', *Theoretical Computer Science* **192**, 107–161.
- BAADER F. AND SIEKMANN J. H. [1994], Unification theory, in D. M. Gabbay, C. J. Hogger and J. A. Robinson, eds, 'Handbook of Logic in Artificial Intelligence and Logic Programming', Oxford University Press, Oxford, UK, pp. 41–125.
- BACHMAIR L. AND GANZINGER H. [2001], Resolution theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19–99.
- BACHMAIR L., GANZINGER H., LYNCH C. AND SNYDER W. [1995], 'Basic paramodulation', *Information and Computation* **121**(2), 172–192.
- BENANAV D., KAPUR D. AND NARENDRA P. [1985], Complexity of matching problems, in J.-P. Jouannaud, ed., 'Proceedings of the 1st International Conference on Rewriting Techniques and Applications', Vol. 202 of *Lecture Notes in Computer Science*, Springer-Verlag, Dijon, France, pp. 417–429.
- BOCKMAYR A. [1992], Algebraic and logical aspects of unification, in K. U. Schulz, ed., 'Proceedings of the 1st International Workshop on Word Equations and Related Topics (IWWERT '90)', Vol. 572 of *Lecture Notes in Computer Science*, Springer-Verlag, Tübingen, Germany, pp. 171–180.
- BOCKMAYR A., KRISCHER S. AND WERNER A. [1992], An optimal narrowing strategy for general canonical systems, in 'Proceedings of the 3rd International Workshop on Conditional and Typed Term Rewriting Systems', Vol. 656 of *Lecture Notes in Computer Science*, Springer-Verlag, Pont à Mousson, France.
- BOUDET A. [1993], 'Combining unification algorithms', *J. Symbolic Computation* **8**, 449–477.
- BOUDET A., CONTEJEAN E. AND DEVIE H. [1990], A new AC-unification algorithm with a new algorithm for solving diophantine equations, in 'Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science', Philadelphia, USA, pp. 141–150.
- BOUDET A., JOUANNAUD J.-P. AND SCHMIDT-SCHAUS M. [1989], 'Unification in Boolean rings and Abelian groups', *J. Symbolic Computation* **8**, 449–477.
- BUNTINE W. L. AND BÜCKERT H.-J. [1994], 'On solving equations and disequations', *J. of the ACM* **41**(4), 591–629.
- BÜCKERT H.-J. [1988], Solving disequations in equational theories, in E. Lusk and R. Overbeek, eds, 'Proceedings of the 9th International Conference on Automated Deduction', Vol. 310 of *Lecture Notes in Computer Science*, Springer-Verlag, Argonne, IL.
- BÜCKERT H.-J. [1989], 'Matching—a special case of unification?', *J. Symbolic Computation* **8**(5), 532–536.
- BÜCKERT H.-J. [1991], *A Resolution Principle for a Logic with Restricted Quantifiers*, Vol. 568 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag.

- BÜRCKERT H.-J., HEROLD A. AND SCHMIDT-SCHAÜSS M. [1989], 'On equational theories, unification, and decidability', *J. Symbolic Computation* 8(3,4), 3–49.
- BURRIS S. AND LAWRENCE J. [1990], 'Unification in commutative rings is not finitary', *Information Processing Letters* 36(1), 37–38.
- BÜTTNER W. [1986a], 'Unification in the data structure multiset', *J. Automated Reasoning* 2(1), 75–88.
- BÜTTNER W. [1986b], Unification in the data structure sets, in J. H. Siekmann, ed., 'Proceedings of the 8th International Conference on Automated Deduction', Vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, Oxford, UK, pp. 470–488.
- BÜTTNER W. [1988], Unification in finite algebras is unitary(?), in E. Lusk and R. Overbeek, eds, 'Proceedings of the 9th International Conference on Automated Deduction', Vol. 310 of *Lecture Notes in Computer Science*, Springer-Verlag, Argonne, IL, pp. 368–377.
- BÜTTNER W., ESTENFELD K., SCHMID R., SCHNEIDER H.-A. AND TIDÉN E. [1990], 'Symbolic constraint handling through unification in finite algebras', *Applicable Algebra in Engineering, Communication and Computing* 1(2), 97–119.
- BÜTTNER W. AND SIMONIS H. [1987], 'Embedding Boolean expressions into logic programming', *J. Symbolic Computation* 4(2), 191–205.
- CHAMPEAUX D. [1986], 'About the Paterson-Wegman linear unification algorithm', *J. Computer and System Sciences* 32, 79–90.
- CLAUSEN M. AND FORTENBACHER A. [1989], 'Efficient solution of linear diophantine equations', *J. Symbolic Computation* 8(1,2), 201–216.
- COHN P. M. [1965], *Universal Algebra*, Harper & Row, New York.
- COLMERAUER A. [1984], Equations and inequations on finite and infinite trees, in 'Proceedings of the International Conference on Fifth Generation Computer Systems', North Holland, Tokyo, Japan, pp. 85–99.
- COMON H. [1988], Unification et Disunification: Théorie et Applications, Ph.D. thesis, Institut National Polytechnique de Grenoble, Grenoble, France.
- COMON H. [1989], Inductive proofs by specification transformation, in N. Dershowitz, ed., 'Proceedings of the 3rd International Conference on Rewriting Techniques and Applications', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Chapel Hill, North Carolina, pp. 76–91.
- COMON H. [1991], Disunification: A survey, in J.-L. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of Alan Robinson', MIT Press, Cambridge, MA, pp. 322–359.
- COMON H. AND LESCANNE P. [1989], 'Equational problems and disunification', *J. Symbolic Computation* 7, 371–425.
- CONTEJEAN E. [1993], 'Solving *-Problems Modulo Distributivity by a Reduction to AC1-Unification', *J. Symbolic Computation* 16(5), 493–521.
- CONTEJEAN E. AND DEVIE H. [1994], 'An efficient incremental algorithm for solving systems of linear diophantine equations', *Information and Computation* 113, 143–172.
- CORBIN J. AND BIDOIT M. [1983], A rehabilitation of Robinson's unification algorithm, in R. E. A. Mason, ed., 'Proceedings of the 9th World Computer Congress, IFIP'83', Elsevier, Paris, France, pp. 909–914.
- DAVIS M. [1973], 'Hilbert's tenth problem is unsolvable', *American Mathematical Monthly* 80, 233–269.
- DEGTYAREV A. AND VORONKOV A. [2001], Equality reasoning in sequent-based calculi, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 10, pp. 609–704.
- DERSHOWITZ N. AND JOUANNAUD J.-P. [1990], Rewrite systems, in J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B: Formal Methods and Semantics, North-Holland, Amsterdam, chapter 6, pp. 243–320.
- DERSHOWITZ N. AND JOUANNAUD J.-P. [1991], 'Notations for rewriting', *Bulletin of the European Association for Theoretical Computer Science* 43, 162–172.

- DEKSHOWITZ N. AND PLAISTED D. [2001], Rewriting, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 9, pp. 533–608.
- DICKSON L. E. [1913], 'Finiteness of the odd perfect and primitive abundant numbers with r distinct prime factors', *Amer. Journal of Math.* **35**, 413–422.
- DOMENJOU D. [1991], Outils pour la déduction automatique dans les théories associatives-commutatives, Thèse de Doctorat, Université de Nancy I, France.
- DOMENJOU D., KLAY F. AND RINGEISEN C. [1994], Combination techniques for non-disjoint equational theories, in A. Bundy, ed., 'Proceedings of the 12th International Conference on Automated Deduction', Vol. 814 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Nancy, France, pp. 267–281.
- DOUGHERTY D. J. AND JOHANN P. [1992], 'An improved general E -unification method', *J. Symbolic Computation* **14**, 303–320.
- DOWEK G. [2001], Higher-order unification and matching, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 16, pp. 1009–1062.
- DWORK C., KANELAKIS P. AND MITCHELL J. C. [1984], 'On the sequential nature of unification', *J. Logic Programming* **1**, 35–50.
- EDER E. [1985], 'Properties of substitutions and unifications', *J. Symbolic Computation* **1**, 31–46.
- FAGES F. [1983], Formes Canoniques dans les Algèbres Booléennes, et Application à la Démonstration Automatique en Logique de Premier Ordre, Thèse de 3ème cycle, Université Paris VI.
- FAGES F. [1984], Associative-commutative unification, in R. E. Shostak, ed., 'Proceedings of the 7th International Conference on Automated Deduction', Vol. 170 of *Lecture Notes in Computer Science*, Springer-Verlag, Napa, USA, pp. 194–208.
- FAGES F. [1987], 'Associative-commutative unification', *J. Symbolic Computation* **3**, 257–275.
- FAGES F. AND HUET G. P. [1983], Complete sets of unifiers and matchers in equational theories, in 'Proceedings of the 5th Colloquium on Automata, Algebra, and Programming', Vol. 159 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 205–220.
- FAGES F. AND HUET G. P. [1986], 'Complete sets of unifiers and matchers in equational theories', *Theoretical Computer Science* **43**(1), 189–200.
- FAY M. [1979], First-order unification in an equational theory, in 'Proceedings 4th Workshop on Automated Deduction', Austin, Texas, pp. 161–167.
- FILGUEIRA M. AND TOMÁS A. P. [1995], 'A fast method for finding the basis of nonnegative solutions to a linear diophantine equation', *J. Symbolic Computation* **19**, 507–526.
- FORTENBACHER A. [1985], An algebraic approach to unification under associativity and commutativity, in J.-P. Jouannaud, ed., 'Proceedings of the 1st International Conference on Rewriting Techniques and Applications', Vol. 202 of *Lecture Notes in Computer Science*, Springer-Verlag, Dijon, France, pp. 381–397.
- FRANZEN M. [1992], 'Hilbert's tenth problem is of unification type zero', *J. Automated Reasoning* **9**, 169–178.
- FRISCH A. M. AND COHN A. G. [1992], An abstract view of sorted unification, in D. Kapur, ed., 'Proceedings of the 11th International Conference on Automated Deduction', Lecture Notes in Artificial Intelligence, Springer-Verlag, Saratoga Springs, NY, USA, pp. 178–192.
- GALLIER J. AND SNYDER W. [1989], 'Complete sets of transformations for general E -unification', *Theoretical Computer Science* **67**(2,3), 203–260.
- GAREY M. R. AND JOHNSON D. S. [1979], *Computers and Intractability. A Guide to the Theory of NP-completeness*, Freeman, New York.
- GHILARDI S. [1997], 'Unification through projectivity', *J. Logic and Computation* **7**(6), 733–752.
- GOGUEN J. A. [1989], What is unification?, in H. Aït-Kaci and M. Nivat, eds, 'Resolution of Equations in Algebraic Structures, Volume 1, Algebraic Techniques', Academic Press, pp. 217–261.
- GRÄTZER G. [1979], *Universal Algebra, Second Edition*, Springer-Verlag, New York.

- GUARD J. R. [1964], Automated logic for semi-automated mathematics, Scientific Report 1, AFCRL 64-411, Air Force Cambridge Lab.
- GUARD J. R. [1969], 'Semi-automated mathematics', *J. of the ACM* **16**(1), 49-62.
- GUO Q., NARENDRA P. AND SHUKLA S. K. [1998], Unification and matching in process algebras, in T. Nipkow, ed., 'Proceedings of the 9th International Conference on Rewriting Techniques and Applications', Vol. 1379 of *Lecture Notes in Computer Science*, Springer-Verlag, Tsukuba, Japan, pp. 91-105.
- GUTIÉRREZ C. [1998], Solvability of word equations is in exponential space, in 'Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science FOCS'98', IEEE Computer Society Press, Palo Alto, California.
- HERBRAND J. [1930a], Recherches sur la Théorie de la Démonstration, Ph.D. thesis, Sorbonne, Paris. Reprinted in W. D. Goldfarb, editor, *Logical Writings*. Reidel, 1971.
- HERBRAND J. [1930b], 'Recherches sur la théorie de la Démonstration', *Travaux de la Société des Sciences et des Lettres de Varsovie, Classe III* **33**(128).
- HERBRAND J. [1967], Investigations in proof theory: The properties of true propositions, in J. van Heijenoort, ed., 'From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931', Harvard University Press, Cambridge, MA, pp. 525-581.
- HERBRAND J. [1971], Recherches sur la théorie de la démonstration, in W. D. Goldfarb, ed., 'Logical Writings', Reidel, Dordrecht.
- HERMANN M. AND KOLAITIS P. G. [1996], Unification algorithms cannot be combined in polynomial time, in M. A. McRobbie and J. K. Slaney, eds, 'Proceedings of the 13th International Conference on Automated Deduction', Vol. 1104 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 246-260.
- HERMANN M. AND KOLAITIS P. G. [1997], On the complexity of unification and disunification in commutative idempotent semigroups, in G. Smolka, ed., 'Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP'97)', Vol. 1330 of *Lecture Notes in Computer Science*, Springer-Verlag, Linz, Austria, pp. 283-297.
- HEROLD A. [1986], Combination of unification algorithms, in J. H. Siekmann, ed., 'Proceedings of the 8th International Conference on Automated Deduction', Vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, Oxford, UK, pp. 450-469.
- HEROLD A. [1987], Combination of Unification Algorithms in Equational Theories, Ph.D. thesis, Universität Kaiserslautern, Kaiserslautern, Germany.
- HEROLD A. AND SIEKMANN J. H. [1987], 'Unification in Abelian semigroups', *J. Automated Reasoning* **3**, 247-283.
- HERRLICH H. AND STRECKER G. E. [1973], *Category Theory*, Allyn and Bacon, Boston.
- HUET G. P. [1976], Résolution d'Équations dans des Langages d'ordre 1,2,..., ω , Thèse d'État, Université de Paris VII.
- HUET G. P. [1978], 'An algorithm to generate the basis of solutions to homogeneous linear diophantine equations', *Information Processing Letters* **7**(3), 144-147.
- HULLOT J.-M. [1980], Canonical forms and unification, in W. Bibel and R. Kowalski, eds, 'Proceedings of the 5th International Conference on Automated Deduction', Vol. 87 of *Lecture Notes in Computer Science*, Springer-Verlag, Les Arcs, France, pp. 318-334.
- ILIOPOULOS C. S. [1989a], 'Worst-case complexity bounds on algorithms for computing the canonical structure of finite Abelian groups and the Hermite and Smith normal forms of an integer matrix', *SIAM J. Computing* **18**(4), 658-669.
- ILIOPOULOS C. S. [1989b], 'Worst-case complexity bounds on algorithms for computing the canonical structure of infinite Abelian groups and solving systems of linear diophantine equations', *SIAM J. Computing* **18**(4), 670-678.
- JAFFAR J. [1990], 'Minimal and complete word unification', *J. of the ACM* **37**(1), 47-85.
- JOUANNAUD J.-P. AND KIRCHNER C. [1991], Solving equations in abstract algebras: A rule-based survey of unification, in J.-L. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of A. Robinson', MIT Press, Cambridge, MA.

- JOUANNAUD J.-P. AND KIRCHNER H. [1986], 'Completion of a set of rules modulo a set of equations', *SIAM J. Computing* 15(4), 1155-1194.
- KANNAN R. AND BACHEM A. [1979], 'Algorithms for computing the Smith and Hermite normal forms of an integer matrix', *SIAM J. Computing* 8(4), 499-507.
- KAPUR D. AND NARENDRA P. [1992a], 'Complexity of unification problems with associative-commutative operators', *J. Automated Reasoning* 9, 261-288.
- KAPUR D. AND NARENDRA P. [1992b], Double exponential complexity of computing complete sets of AC-unifiers, in 'Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science', Santa Cruz, California, pp. 11-21.
- KEPSER S. [1999], Negation in combining constraint systems, in D. Gabbay and M. de Rijke, eds, 'Frontiers of Combining Systems 2, Papers presented at FroCoS'98', Research Studies Press/Wiley, Amsterdam, pp. 117-192.
- KEPSER S. AND RIGHTS J. [1999], Optimisation techniques for combining constraint solvers, in D. Gabbay and M. de Rijke, eds, 'Frontiers of Combining Systems 2, Papers presented at FroCoS'98', Research Studies Press/Wiley, Amsterdam, pp. 193-210.
- KFOURY A., TIURYN J. AND URZYCZYN P. [1993], 'The undecidability of the semi-unification problem', *Information and Computation* 102(1), 83-101.
- KIRCHNER C. [1985], Méthodes et Outils de Conception Systématique d'Algorithmes d'Unification dans les Théories Equationnelles., Thèse d'État, Université de Nancy I, France.
- KIRCHNER C. AND KIRCHNER H. [1989], Constrained equational reasoning, in 'Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation', ACM Press, Portland, Oregon.
- KIRCHNER C. AND LESCANNE P. [1987], Solving disequations, in 'Proceedings of the Annual IEEE Symposium on Logic in Computer Science', Ithaca, NY, pp. 347-352.
- KIRCHNER H. AND RINGEISSEN C. [1994], 'Combining symbolic constraint solvers on algebraic domains', *J. Symbolic Computation* 18(2), 113-155.
- KNUTH D. E. [1981], *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Computer Science and Information Processing, second edn, Addison-Wesley, Reading.
- KNUTH D. E. AND BENDIX P. B. [1970], Simple word problems in universal algebras, in J. Leech, ed., 'Computational Problems in Abstract Algebra', Pergamon Press, Oxford.
- KOSCIELSKI A. AND PACHOLSKI L. [1990], Complexity of unification in free groups and free semigroups, in 'Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science', Los Alamitos, pp. 824-829.
- KOZEN D. [1981], 'Positive first-order logic is NP-complete', *IBM Journal for Research and Development* 25, 327-332.
- LAMBERT J.-L. [1987], 'Une borne pour les générateurs des solutions entières positives d'une équation diophantienne linéaire', *Comptes Rendus de l'Académie des Sciences de Paris* 305, 39-40.
- LANKFORD D. S., BUTLER G. AND BRADY B. [1984], 'Abelian group unification algorithms for elementary terms', *Contemporary Mathematics* 29, 193-199.
- LASSEZ J.-L., MAHER M. AND MARIOTT K. [1987], Unification revisited, in J. Minker, ed., 'Foundations of Deductive Databases and Logic Programming', Morgan Kaufman, Los Altos, California, pp. 587-625.
- LINCOLN P. AND CHRISTIAN J. [1989], 'Adventures in associative-commutative unification', *J. Symbolic Computation* 8(1,2), 217-240.
- LIVESEY M. AND SIEKMANN J. H. [1975], Unification of AC-terms (bags) and ACI-terms (sets), Internal report, University of Essex. Also published as Technical Report 3-76, Universität Karlsruhe, 1976.
- MAKANIN G. S. [1977], 'The problem of solvability of equations in a free semigroup', *Math. Sbornik* 103, 147-236. English translation in *Math. USSR Sbornik* 32, 1977.
- MAL'CEV A. I. [1971], *The Metamathematics of Algebraic Systems*, Vol. 66 of *Studies in Logic and the Foundation of Mathematics*, North Holland, Amsterdam.

- MAL'CEV A. I. [1973], *Algebraic Systems*, Vol. 192 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen*, Springer-Verlag, Berlin.
- MARTELLI A. AND MONTANARI U. [1976], Unification in linear time and space: A structured presentation., Technical Report B76-16, University of Pisa.
- MARTELLI A. AND MONTANARI U. [1982], 'An efficient unification algorithm', *ACM Transactions on Programming Languages and Systems* 4(2), 258-282.
- MARTIN U. AND NIPKOW T. [1989a], 'Boolean unification—The story so far', *J. Symbolic Computation* 7(3,4), 275-293.
- MARTIN U. AND NIPKOW T. [1989b], 'Unification in Boolean rings', *J. Automated Reasoning* 4, 381-396.
- MATYASEVICH Y. [1971], 'Diophantine representation of recursively enumerable predicates', *Ak. Nauk USSR, Ser. Math.* 35, 3-30.
- MESEGUER J., GOGUEN J. A. AND SMOLKA G. [1989], 'Order-sorted unification', *J. Symbolic Computation* 8, 383-413.
- MOSER M. [1993], Improving transformation systems for general E -unification, in C. Kirchner, ed., 'Proceedings of the 5th International Conference on Rewriting Techniques and Applications', Lecture Notes in Artificial Intelligence, Springer-Verlag, Montreal, Canada, pp. 92-105.
- NARENDHAN P. [1996a], Solving linear equations over polynomial semirings, in 'Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, New Brunswick, New Jersey, pp. 466-472.
- NARENDHAN P. [1996b], 'Unification modulo $AC1+1+0$ ', *Fundamenta Informaticae* 25(1), 49-57.
- NARENDHAN P. AND OTTO F. [1990], Some results on equational unification, in M. E. Stickel, ed., 'Proceedings of the 10th International Conference on Automated Deduction', Vol. 449 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Kaiserslautern, Germany, pp. 276-291.
- NIJEUWENHUIS R. AND RUBIO A. [1994], AC-superposition with constraints: No AC-unifiers needed, in A. Bundy, ed., 'Proceedings of the 12th International Conference on Automated Deduction', Vol. 814 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Nancy, France, pp. 545-559.
- NIJEUWENHUIS R. AND RUBIO A. [2001], Paramodulation-based theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 7, pp. 371-443.
- NIPKOW T. [1990], 'Unification in primal algebras, their powers and their varieties', *J. of the ACM* 37(1), 742-776.
- NUTT W. [1990], Unification in monoidal theories, in M. E. Stickel, ed., 'Proceedings of the 10th International Conference on Automated Deduction', Vol. 449 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Kaiserslautern, Germany, pp. 618-632.
- NUTT W. [1991], 'The unification hierarchy is undecidable', *J. Automated Reasoning* 7(3), 369-381.
- NUTT W., RÉTY P. AND SMOLKA G. [1989], 'Basic narrowing revisited', *J. Symbolic Computation* 7(3,4), 295-317.
- OLIART A. AND SNYDER W. [1998], A fast algorithm for uniform semiunification, in H. Kirchner and C. Kirchner, eds, 'Proceedings of the 15th International Conference on Automated Deduction', Vol. 1421 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Lindau, Germany, pp. 239-253.
- PATERSON M. S. AND WEGMAN M. N. [1978], 'Linear unification', *J. Computer and System Sciences* 16(2), 158-167.
- PÉCUCHE J. P. [1981], Équations avec constantes et algorithme de Makanin, Thèse de doctorat, Laboratoire d'Informatique, University of Rouen.
- PETERSON G. [1983], 'A technique for establishing completeness results in theorem proving with equality', *SIAM J. Computing* 12(1), 82-100.

- PETERSON G. AND STICKEL M. E. [1981], 'Complete sets of reductions for equational theories with complete unification algorithms', *J. of the ACM* **28**(2), 233–264.
- PIERCE B. C. [1991], *Basic Category Theory for Computer Scientists*, MIT Press, Cambridge, Mass.
- PLANDOWSKI W. [1999a], Satisfiability of word equations with constants is in NEXPTIME, in T. Leighton, ed., 'Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC'99)', ACM Press, Atlanta, Georgia.
- PLANDOWSKI W. [1999b], Satisfiability of word equations with constants is in PSPACE, in P. Beame, ed., 'Proceedings of the Fortieth Annual IEEE Symposium on Foundations of Computer Science (FOCS'99)', IEEE Computer Society Press, New York City, NY.
- PLOTKIN G. [1972], 'Building in equational theories', *Machine Intelligence* **7**, 73–90.
- POTTIER L. [1991], Minimal solutions of linear diophantine equations: Bounds and algorithms, in R. V. Book, ed., 'Proceedings of the 4th International Conference on Rewriting Techniques and Applications', Vol. 488 of *Lecture Notes in Computer Science*, Springer-Verlag, Como, Italy, pp. 162–173.
- PRAWITZ D. [1960], 'An improved proof procedure', *Theoria* **26**, 102–139.
- RÉTY P. [1987], Improving basic narrowing techniques, in P. Lescanne, ed., 'Proceedings of the 2nd International Conference on Rewriting Techniques and Applications', Vol. 256 of *Lecture Notes in Computer Science*, Springer-Verlag, Bordeaux, France, pp. 216–227.
- RINGEISSEN C. [1992], Unification in a combination of equational theories with shared constants and its application to primal algebras, in A. Voronkov, ed., 'Proceedings of the Conference on Logic Programming and Automated Reasoning', *Lecture Notes in Artificial Intelligence*, Springer-Verlag, St. Petersburg, Russia, pp. 261–272.
- ROBINSON J. A. [1963], 'Theorem proving on the computer', *J. of the ACM* **10**(2), 163–174.
- ROBINSON J. A. [1965], 'A machine oriented logic based on the resolution principle', *J. of the ACM* **12**(1), 23–41.
- ROBINSON J. A. [1971], 'Computational logic: The unification computation', *Machine Intelligence* **6**, 63–72.
- RYDEHEARD D. E. AND BURSTALL R. M. [1985], A categorical unification algorithm, in 'Proceedings of the Workshop on Category Theory and Computer Programming', Vol. 240 of *Lecture Notes in Computer Science*, Springer-Verlag, Guildford, UK, pp. 493–505.
- SCHMIDT R. A. [1998], *E*-unification for subsystems of S4, in T. Nipkow, ed., 'Proceedings of the 9th International Conference on Rewriting Techniques and Applications', Vol. 1379 of *Lecture Notes in Computer Science*, Springer-Verlag, Tsukuba, Japan, pp. 106–120.
- SCHMIDT-SCHAUS M. [1986a], Unification in many-sorted equational theories, in 'Proceedings of the 8th International Conference on Automated Deduction', Vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 538–552.
- SCHMIDT-SCHAUS M. [1986b], 'Unification under associativity and idempotence is of type nullary', *J. Automated Reasoning* **2**(3), 277–282.
- SCHMIDT-SCHAUS M. [1989a], *Computational Aspects of an Order Sorted Logic With Term Declarations*, Vol. 395 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag.
- SCHMIDT-SCHAUS M. [1989b], 'Unification in a combination of arbitrary disjoint equational theories', *J. Symbolic Computation* **8**(1,2), 51–99.
- SCHMIDT-SCHAUS M. [1992], Some results for unification in distributive equational theories, Internal Report 7/92, Universität Frankfurt, Frankfurt, Germany.
- SCHMIDT-SCHAUS M. [1996a], An algorithm for distributive unification, in H. Ganzinger, ed., 'Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA-96)', Vol. 1103 of *Lecture Notes in Computer Science*, Springer-Verlag, New Brunswick, NJ, USA, pp. 287–301.
- SCHMIDT-SCHAUS M. [1996b], 'Decidability of unification in the theory of one-sided distributivity and a multiplicative unit', *Journal of Symbolic Computation* **22**(3), 315–344.

- SCHULZ K. U. [1992], Makanin's algorithm for word equations: Two improvements and a generalization, in K. U. Schulz, ed., 'Proceedings of the 1st International Workshop on Word Equations and Related Topics (IWWERT '90)', Vol. 572 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 85–150.
- SCHULZ K. U. [1993], 'Word unification and transformation of generalized equations', *J. Automated Reasoning* 11, 149–184.
- SCHULZ K. U. [1997], A criterion for intractability of E -unification with free function symbols and its relevance for combination of unification algorithms, in H. Comon, ed., 'Proceedings of the 8th International Conference on Rewriting Techniques and Applications', Vol. 1232 of *Lecture Notes in Computer Science*, Sitges, Spain, pp. 284–307.
- SCHULZ K. U. [1999], 'Tractable and intractable instances of combination problems for unification and disunification', *J. Logic and Computation*. To appear.
- SIEKMANN J. H. [1979], Unification of commutative terms, in 'Proceedings of the International Symposium on Symbolic and Algebraic Manipulation, EUROSAM'79', Vol. 72 of *Lecture Notes in Computer Science*, Springer-Verlag, Marseille, France, pp. 531–545.
- SIEKMANN J. H. [1989], 'Unification theory: A survey', *J. Symbolic Computation* 7(3,4), 207–274.
- SIEKMANN J. H. AND SZABÓ P. [1989], 'The undecidability of the D_A -unification problem', *J. Symbolic Computation* 54(2), 402–414.
- SNYDER W. [1991], *A Proof Theory for General Unification*, Birkhäuser, Boston, Basel, Berlin.
- SOCHER-AMBROSIOUS R. [1994], A refined version of general E -unification, in A. Bundy, ed., 'Proceedings of the 12th International Conference on Automated Deduction', Vol. 814 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Nancy, France, pp. 665–677.
- STICKEL M. E. [1975], A complete unification algorithm for associative-commutative functions, in 'Proceedings of the 4th International Joint Conference on Artificial Intelligence', Tblisi, USSR, pp. 71–82.
- STICKEL M. E. [1981], 'A unification algorithm for associative commutative functions', *J. of the ACM* 28(3), 423–434.
- SZABÓ P. [1982], Unifikationstheorie Erster Ordnung, Ph.D. thesis, Universität Karlsruhe. In German.
- TIDÉN E. [1986], Unification in combinations of collapse-free theories with disjoint sets of function symbols, in J. H. Siekmann, ed., 'Proceedings of the 8th International Conference on Automated Deduction', Vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, Oxford, UK, pp. 431–449.
- TIDÉN E. AND ARNBORG S. [1987], 'Unification problems with one-sided distributivity', *J. Symbolic Computation* 3(1–2), 183–202.
- TOMMASI M. [1991], Automates avec tests d'égalités entre cousins germains, Master's thesis, Université de Lille, France.
- VENTURINI-ZILLI M. [1975], 'Complexity of the unification algorithm for first order expressions', *Calcolo* 12(4), 361–371.
- VOGEL E. [1978], Unification von Morphismen, Diploma thesis, Univ. Karlsruhe, Institut für Informatik I, Karlsruhe, Germany. In German.
- WALTHER C. [1983], A many-sorted calculus based on resolution and paramodulation, in A. Bundy, ed., 'Proceedings of the 8th International Joint Conference on Artificial Intelligence', Vol. 2, Karlsruhe, Germany, pp. 882–891.
- WALTHER C. [1987], *A Many-Sorted Calculus Based on Resolution and Paramodulation*, Research Notes in Artificial Intelligence, Pitman Ltd.
- WEIDENBACH C. [1996], 'Unification in sort theories and its applications', *Annals of Mathematics and Artificial Intelligence* 18(2–4), 261–293.
- WEIDENBACH C. [1998], Sorted unification and tree automata, in W. Bibel and P. Schmidt, eds, 'Automated Deduction – A Basis for Applications, Vol. I: Foundations – Calculi and Methods', Vol. 8 of *Applied Logic Series*, Kluwer Academic Publishers, Dordrecht, NL, pp. 291–320.

- YELICK K. [1987], 'Unification in combinations of collapse-free regular theories', *J. Symbolic Computation* **3**(1,2), 153-182.

Index

Symbols

$[\sigma]$	454
\perp	495

A

A	482
Abelian group	486
AC	484
ACI	485
ACU	484, 504
ACUI	485, 504
ACUZI	485
AG	486, 504
alien subterm	515
associativity	482
associativity-commutativity	484
associativity-commutativity- idempotency	485
at least infinitary	471
at most finitary	471

B

B	496
termination of	456
basic	495
basic rewrite sequence	490
blocking rules	503
Boolean algebra	504
Boolean ring	486, 504
Boolean semiring BS	506
BR	486
BS	506

C

C	483
$C_{\mathcal{F}}(E)$	481
coequalizer	481
collapse-free theory	514
combination problem	513
commutative ring	486
commutative theory	509
commutativity	483
complete set	475
complete set of E -unifiers	470
completeness	457, 463, 489
complexity of unification	453
CRU	486

D

D	484
---------	-----

decision procedure for E -unification ..	472
decomposed form	515
distributivity	484
disunification	519, 520
D^l	484
D^r	484

E

E^ω	492
eager reduce strategy	501
eager rule application	500
E -free algebra	477
elementary E -unification problem	474
End	486
endomorphisms	486
equal modulo E	469
equational class	477
equational theory	469
non-trivial	477
trivial	477
equational unification	448, 469
E -unifiable	470
E -unification	448
E -unification algorithm	472
minimal	472
E -unification problem	469
elementary	474
general	474
with constants	474
with lcr	479
with linear constant restrictions ..	479
E -unification procedure	472
minimal	472
E -unifier	469
most general	470

F

\mathcal{F} -term	451
finitary	471
at most	471
free algebra	477
free amalgamated product	519

G

G	488
general E -unification problem	474
$Gr^{\succ}(E)$	490

H

hereditary restrictions	495
-------------------------------	-----

higher-order unification448

I

idempotency 485
 idempotent constant509
 identity 469
 implicit operation509
 index of a rule496, 498, 503
 infinitary471
 at least471
 inhomogeneous linear equation 507
 instantiation 447
 instantiation quasi-ordering452
 instantiation quasi-ordering modulo E 470,
 473

L

labelling function516
 lazy paramodulation488
 lcr479
 left distributivity484
 linear constant restrictions479
 linear equation in \mathcal{BS} 506
 linear equation in \mathcal{N} 506
 linear equation in semiring 504
 linear equation in \mathcal{Z} 506
 linear ordering 516

M

matching519
 matrix over S_E 510
 mgu 452, 453, 455, 457–459, 464, 465
 equivalent458
 idempotent458
 properties of458
 uniqueness of458
 minimal complete set475
 minimal complete set of E -unifiers470
 minimal E -unification algorithm472
 minimal E -unification procedure472
 monoidal theory508
 more general
 modulo E 470
 most general E -unifier470
 most general unifier447, 452
 in category481
 modulo E 470
 multiset452

N

\mathcal{N} 498
 narrowing 488, 495
 basic496, 499
 inference rule for499, 503

standard498
 non-trivial equational theory 477

O

occurs check 455, 457, 462, 463
 oriented ground instance 490

P

partition516
 Plotkin's procedure472, 483
 positive AE fragment477
 positive AE sentence 477
 positive AE theory 478
 positive existential fragment477
 positive existential sentence 477
 positive existential theory 478
 positive fragment477
 positive sentence 477
 positive theory 478
 primal algebra504
 pure unification problem515

R

R_E 490
 redex orderings501
 reduction ordering 490, 496, 497
 regular theory514
 rewrite proof488
 right distributivity484
 ring of integers506

S

S 496, 500
 σ_S 454
 schema term464
 S_E 510
 semantic approach487, 503
 semi-additive category509
 semiring504
 corresponding to E 510
 semiring \mathcal{N} 506
 semiunification519
 set of identities469
 shared variable516
 $Sig(E)$ 469
 signature451
 simplification499
 solved form454
 sorted unification520
 soundness457, 463
 structure sharing460
 substitution451
 idempotent452, 454, 456, 458
 more general452

reduced	489
triangular form	452
syntactic approach	487
syntactic unification	447
system	
constraint	495
of equations	454

T

$\mathcal{T}(\mathcal{F}, \nu)$	451
term dag	
definition of	460
substitution as relation on	460
unification of	459
trivial equational theory	477
type zero	471

U

U	484
\mathcal{U}	454, 492, 495
correctness	457
solutions from	456
$\mathcal{U}_E(\Gamma)$	470
unification algorithms	
almost linear	463
complexity	453, 459, 463, 468
correctness	453, 457, 466
implementation	453
naive	452
recursive descent on dags	461
recursive descent on trees	453
rule-based	454
unification closure	464
unification modulo E	448
unification problem	452
in a category	481
unification type	471
w.r.t. elementary unification	474
w.r.t. general unification	474
w.r.t. unification with constants ..	474
unifier	447, 452
in category	481
most general	447, 452
unitary	471

V

$V(E)$	477
variable abstraction	501
variable elimination 455, 457, 495, 498, 501	
basic	496
variable identification	516
variable renaming	452
variety	477
$\mathcal{Vars}(t)$	451

Z

\mathcal{Z}	506
---------------------	-----

Rewriting

Nachum Dershowitz

David A. Plaisted

SECOND READERS: Bernhard Gramlich, Mitch Harris, and Konstantin Korovin.

Contents

1	Introduction	537
2	Terminology	541
3	Normal Forms and Validity	544
4	Termination Properties	546
5	Church-Rosser Properties	559
6	Completion	567
7	Relativized Rewriting	574
8	Equational Theorem Proving	581
9	Conditional Rewriting	585
10	Programming	593
	Bibliography	597
	Index	608

If the work entailed amounts to a virtual rewriting, the resulting typescript or manuscript is a re-write.

—Eric Partridge, *Slang To-day & Yesterday* (1933)

1. Introduction

Equations lie at the foundation of mathematics and the sciences. Sometimes one needs to determine if an identity follows logically from axioms; other times, one seeks solutions to an equation; oftentimes one wishes to compute an equivalent, simpler form for a given expression. These equational reasoning abilities are indispensable in many computer applications, including symbolic algebraic computation, automated deduction, program specification and verification, and high-level programming languages and environments.

Rewriting is a very powerful method for dealing computationally with equations. Oriented equations, called *rewrite rules*, are used to replace equals by equals, but only in one direction. The theory of rewriting centers around the concept of *normal form*, an expression that cannot be rewritten any further. Computation consists of rewriting to a normal form; when the normal form is unique, it is taken as the value of the initial expression. When rewriting equal terms always leads to the same normal form, the set of rules is said to be *convergent* and rewriting can be used to decide validity of identities in the equational theory. Rewriting has the computational power of Markov algorithms—and of recursive functions and Turing machines. The basic ideas hark back to Axel Thue [1914].

Within automated theorem provers, derived equations can be used freely for *simplification* provided progress toward normal forms is guaranteed. This process is sometimes called *demodulation*. The trick is to ensure that one can delete pre-simplified formulæ without compromising completeness of the prover.

We begin with several motivating examples.

1.1. EXAMPLE (*Hercules and Hydra*). Hydra is a bush-like creature with multiple heads attached by stems to the ends of branches. Branches have nodes with buds along their length, and may ramify at any node. Each time Hercules hacks off a head of Hydra's, one of the nodes along the branch leading to that head sprouts some number of new branches identical to—and adjacent to—the weakened branch that previously supported the severed head, together with all its remaining heads and nodes. Furthermore, when any node loses all its branches and stems, its bud ripens into a new head. But when the root loses a stem with a head, nothing is regenerated.

Suppose Hydra starts off as a lone stalk with 10 buds along its length and one head at the apex. Chopping that head results in the original stalk topped by the stump of a head, which ripens, leaving a stalk of length 9 with 1 head. Then one of the buds along the stalk sprouts many copies of the branch above it, together with its fresh head. Chopping off one of these new heads again causes a bud below

the chopped head to regenerate a number of copies of the branch supporting the severed head, together with its heads, buds, and branches.

If ever Hercules eliminates all of Hydra's heads, Hydra loses. The question is: How can Hercules defeat Hydra?

Establishing termination of rewriting processes is the topic of Section 4.

1.2. EXAMPLE (*Grecian Urn*). An urn holds 150 black beans and 75 white. Two beans are removed at a time: if they're the same color, a black one is placed in the urn; if they're different, the white one is returned. The process is repeated as long as possible. This problem may be expressed as the following system of replacement rules:

$$\begin{array}{llll} BB & \rightarrow & B & \quad WW \rightarrow B \\ WB & \rightarrow & W & \quad BW \rightarrow W \\ BW & \rightarrow & WB & \quad WB \rightarrow BW \end{array}$$

the bottom two of which indicate that beans shift around in the urn. Is the color of the last bean in the urn predetermined and, if so, what is it?

Determinism of output is covered in Section 5. How to handle permutative rules such as the last two is the subject of Section 7. In Section 6, *ordered* rewriting, using rewrite relations that are defined in terms of a set of unordered equations plus a well-founded partial ordering, is considered.

1.3. EXAMPLE (*Chameleon Island*). The chameleons on this strange island come in three colors, red, yellow, and green, and meander about continuously. Whenever two chameleons of differing colors confront each other, they both change to the neutral third color. This setup can be expressed by six rules:

$$\begin{array}{llll} RY & \rightarrow & GG & \quad YR \rightarrow GG \\ GY & \rightarrow & RR & \quad YG \rightarrow RR \\ RG & \rightarrow & YY & \quad GR \rightarrow YY \end{array}$$

Suppose there are initially 15 red chameleons, 14 yellow, and 13 green. Can their haphazard meetings lead to a stable, monochromatic state?

Analyzing the interaction of rules is the topic of Section 6.

1.4. EXAMPLE (*Insertion Sort*). As a more prosaic example of rewrite system, consider the following program to rearrange a list of natural numbers in non-increasing order by inserting elements one-by-one into position:

$$\begin{array}{llll} \max(0, x) & \rightarrow & x & \\ \max(x, 0) & \rightarrow & x & \quad \max(s(x), s(y)) \rightarrow s(\max(x, y)) \\ \min(0, x) & \rightarrow & 0 & \\ \min(x, 0) & \rightarrow & 0 & \quad \min(s(x), s(y)) \rightarrow s(\min(x, y)) \\ \text{sort}(\epsilon) & \rightarrow & \epsilon & \quad \text{sort}(x : y) \rightarrow \text{insert}(x, \text{sort}(y)) \\ \text{insert}(x, \epsilon) & \rightarrow & x : \epsilon & \quad \text{insert}(x, y : z) \rightarrow \max(x, y) : \text{insert}(\min(x, y), z) \end{array}$$

Lists are represented as nested pairs [e.g. $\langle 3, 1, 4, 1 \rangle$ as $3 : (1 : (4 : (1 : \epsilon)))$] and numbers in tally (unary) notation [e.g. 4 is short for $s(s(s(s(0))))$]. Computation proceeds by computing the normal form of an input term of the form $\text{sort}(3 : 1 : 4 : 1 : \epsilon)$. We would like to ascertain that every well-sorted ground term constructed from sort , $:$, ϵ , s , and 0 is equal to a unique term not containing sort , nor the auxiliary symbols, insert , max , and min . In addition, one might wish to prove inductive properties of the program, such as $\text{sort}(\text{sort}(\ell)) = \text{sort}(\ell)$ for all lists ℓ of natural numbers.

This is an example of an *orthogonal* rewrite system, defined in Section 5 and studied in Section 10. For computation, rewrite rules are usually applied nondeterministically, since, in general, more than one rule can be applied, and any one rule may apply at more than one position within a term. Regarding proofs of inductive properties, see [Comon 2001] (Chapter 14 of this Handbook).

1.5. EXAMPLE (*Loops*). Consider the following dozen rules:

$$\begin{array}{ll} x \backslash x & \rightarrow e & x \cdot (x \backslash y) & \rightarrow y \\ x / x & \rightarrow e & (y / x) \cdot x & \rightarrow y \\ e \cdot x & \rightarrow x & x \backslash (x \cdot y) & \rightarrow y \\ x \cdot e & \rightarrow x & (y \cdot x) / x & \rightarrow y \\ e \backslash x & \rightarrow x & x / (y \backslash x) & \rightarrow y \\ x / e & \rightarrow x & (x / y) \backslash x & \rightarrow y \end{array}$$

Each rule follows by algebraic manipulation from some combination of the following seven axioms for algebraic structures called *loops*, which are groups without associativity:

$$\begin{array}{ll} x \cdot (x \backslash y) & = y & (y / x) \cdot x & = y \\ x \backslash (x \cdot y) & = y & (x / y) \backslash x & = y \\ x \backslash (x \cdot y) & = y & (y \cdot x) / x & = y \\ y / y & = x \backslash x \end{array}$$

along with the defining equation

$$y / y = e$$

Rewrite systems were designed to be used as decision procedures for truth of an equation in all models of the theory. To decide whether an arbitrary equation is a valid identity for all loops—in which case it can be proved by purely equational reasoning, we need to ascertain that *any* two terms equal in the theory have the same normal forms. Section 3 deals with deciding validity by rewriting. Constructing such systems is the subject of Section 6.

1.6. EXAMPLE (*Interpreter*). Rewrite systems can be used to interpret other programming languages. The machine state of a Turing-equivalent two-counter device

$$\begin{array}{llll}
eval(\mathbf{zap0}, \langle x, y \rangle) & \rightarrow & \langle 0, y \rangle & eval(\mathbf{zap1}, \langle x, y \rangle) \rightarrow \langle x, 0 \rangle \\
eval(\mathbf{inc0}, \langle x, y \rangle) & \rightarrow & \langle s(x), y \rangle & eval(\mathbf{inc1}, \langle x, y \rangle) \rightarrow \langle x, s(y) \rangle \\
eval(\mathbf{dec0}, \langle 0, y \rangle) & \rightarrow & \langle 0, y \rangle & eval(\mathbf{dec1}, \langle x, 0 \rangle) \rightarrow \langle x, 0 \rangle \\
eval(\mathbf{dec0}, \langle s(x), y \rangle) & \rightarrow & \langle x, y \rangle & eval(\mathbf{dec1}, \langle x, s(y) \rangle) \rightarrow \langle x, y \rangle \\
eval(\mathbf{ifpos0} \ p, \langle 0, y \rangle) & \rightarrow & \langle 0, y \rangle & eval(\mathbf{ifpos1} \ p, \langle x, 0 \rangle) \rightarrow \langle x, 0 \rangle \\
eval(\mathbf{ifpos0} \ p, \langle s(x), y \rangle) & \rightarrow & eval(p, \langle s(x), y \rangle) & \\
& & eval(\mathbf{ifpos1} \ p, \langle x, s(y) \rangle) & \rightarrow eval(p, \langle x, s(y) \rangle) \\
\mathbf{whilepos0} \ p & \rightarrow & (\mathbf{ifpos0} \ p; \mathbf{whilepos0} \ p) & \\
& & \mathbf{whilepos1} \ p & \rightarrow (\mathbf{ifpos1} \ p; \mathbf{whilepos1} \ p) \\
eval((p; q), u) & \rightarrow & eval(q, eval(p, u)) &
\end{array}$$

Figure 1: Two-counter machine interpreter.

can be represented as a pair $\langle x, y \rangle$. The semantics of its instruction set can be defined by the rules for an interpreter, shown in Fig. 1. To simulate a machine computation, one rewrites a term of the form

$$eval(\text{program}, \langle \text{input}, 0 \rangle)$$

until a normal form is reached. The penultimate rule, for example, can clearly be applied *ad infinitum*. A specific strategy of rule application is needed to guarantee that a normal form is attained whenever one exists.

Programming issues are discussed in Section 10.

1.7. EXAMPLE (*Stack*). In some cases, it is convenient to attach conditions to rules. For example, stack operations can be implemented as follows:

$$\begin{array}{llll}
top(push(x, y)) & \rightarrow & x & pop(push(x, y)) \rightarrow y \\
empty?(e) & \rightarrow & T & empty?(push(x, y)) \rightarrow F \\
empty?(x) = F & | & push(top(x), pop(x)) & \rightarrow x
\end{array}$$

The condition of the last rule ensures that only nonempty stacks are popped. Conditional rewriting and related inference techniques are the subject of Chapter 9.

Semi-Thue systems, or string-rewriting systems, are a related formalism of equivalent computational strength. They originated historically in an attempt to investigate computability and can be simulated by rewrite systems in which every function symbol is monadic (has arity 1). Such systems have an implicit variable at the right end; for example, $f(g(f(g(x))))$, though written as *fgffg*, rewrites any suffix *fgffgx* having prefix *fgffg*. For such systems, rewriting corresponds to substring replacement. Another way of viewing such systems is as presentations of monoids, where

there are a finite number of individual constants and an associative concatenation operator. Semi-Thue systems will not be further considered here. The monographs [Benninghofen, Kemmerich and Richter 1987, Book and Otto 1993] describe methods and results for strings.

Several other important topics in rewriting are only touched upon (modularity, for example), or not covered at all (higher-order rewriting and “sequentiality”, for example), in this chapter, which concentrates on the use of rewriting techniques in the context of automated deduction. Other surveys of rewriting include [Avenhaus and Madlener 1990, Dershowitz and Jouannaud 1990, Klop 1992, Plaisted 1993, Avenhaus and Madlener 1989, Baader and Nipkow 1998].

2. Terminology

The central idea of rewriting is to impose directionality on the use of equations in proofs. A *rewrite rule* is an ordered pair, written $l \rightarrow r$, of terms l and r . Like equations, rules are used to replace instances of l by corresponding instances of r ; unlike equations, rules are not used to replace instances of the right-hand side r .

A (*free, first-order*) *term* over symbols \mathcal{F} , constants \mathcal{C} , and variables \mathcal{X} is either a variable $x \in \mathcal{X}$, an individual constant $c \in \mathcal{C}$, or an expression of the form $f(t_1, t_2, \dots, t_n)$, where $f \in \mathcal{F}$ is a function symbol of arity (number of arguments) n and the t_i are terms.¹ Each function symbol has one or more *arities*, which are nonnegative integers indicating how many arguments it can take. A symbol allowing an arbitrary number of arguments is called *variadic*. For example, $f(a, g(x, y))$ is a term, with *outermost* function symbol f . Terms can be seen as labeled rooted ordered trees. Let the set of all such terms be denoted \mathcal{T} . A term t is *linear* if each variable appears at most once in t . Thus the term $f(x, x, z)$ is not linear. A *ground term* is a term containing no variables. To avoid confusion, we will use the symbol \equiv for syntactic identity of terms.

A (*strict*) *partial ordering* $>$ is an irreflexive transitive relation. A *quasi-ordering* \succeq is a reflexive transitive relation. We try to use the sign $>$ for partial orderings and \succeq for quasi-orderings, where convenient. As usual, $y < x$ means the same as $x > y$, $x \geq y$ means either $x > y$ or $x = y$, and $x \succ y$ means $x \succeq y$ but not $y \succeq x$. The relation \succ is called the *strict part* of \succeq and is a partial ordering. If both $x \succeq y$ and $y \succeq x$, we say that x and y are equivalent (though they may be distinct) and write $x \approx y$.

We say that a term u is a *subterm* of t and write $t \geq_{\text{sub}} u$ if either $t \equiv u$ or if $t \equiv f(t_1, \dots, t_n)$ and u is a subterm of t_i for some i . We write $t >_{\text{sub}} u$ to indicate that u is a proper subterm of t . When needed, it is customary to specify positions of subterms as sequences of integers indicating the path to the subterm in the tree representation of the term. If α is a position and t is a term, we would write $t|_{\alpha}$ for the subterm of t at position α , defined by $t|_{\epsilon} \equiv t$ for the empty sequence (top position) ϵ ; otherwise, $f(t_1, \dots, t_n)|_{i:\alpha} \equiv t_i|_{\alpha}$. A *context* is a term with some subterms replaced by *holes*, here signified by \diamond . A lone \diamond is thus the

¹In this chapter, we consider only unsorted terms, for simplicity.

simplest context. If t is a context with m holes (an m -context) and s_1, \dots, s_m are terms, then $t[s_1, \dots, s_m]$ indicates t with the \diamond 's replaced by s_1, \dots, s_m in a predetermined order. At the same time, $t[s]$ indicates that the term t contains an occurrence of the subterm s . Two occurrences of subterms are *disjoint* if neither is a subterm of the other.

A *substitution* is a partial mapping from variables to terms (or contexts) denoted by $\{x_1 \mapsto s_1, x_2 \mapsto s_2, \dots, x_n \mapsto s_n\}$, indicating that the variable x_i maps to the term s_i . (Though traditional, there is usually no need to restrict the domain of a substitution to be a finite subset of the variables.) Substitutions $\sigma : \mathcal{X} \rightarrow \mathcal{T}$ are extended to a total mapping $\sigma : \mathcal{T} \rightarrow \mathcal{T}$ of (first-order) contexts; if t is a term or a context and σ is a substitution, we use postfix notation $t\sigma$ for the image of t under σ . This is defined as follows: If $\sigma : x_i \mapsto s_i$, then $x_i\sigma \equiv s_i$; if x is a variable not in the domain of σ , then $x\sigma \equiv x$; if $f(t_1, \dots, t_k)$ is a nonvariable term then $f(t_1, \dots, t_k)\sigma \equiv f(t_1\sigma, \dots, t_k\sigma)$. Also, $\diamond\sigma \equiv \diamond$. The image of an equation, rewrite rule, or formula under a substitution is defined similarly. If t is a term, $t\sigma$ is called an *instance* of t . Note that $t[u]\sigma \equiv (t\sigma)[u\sigma]$. A term r is a (*renamed*) *variant* of s if they are instances of each other. If $t\sigma$ is a ground term, we call σ a *ground substitution* for t and $t\sigma$, a *ground instance* of t . Similar terminology applies to instances of equations, rewrite rules, and formulæ.

A *term-rewriting* or *rewrite system* R is a set of *rewrite rules*, $l \rightarrow r$, where l and r are each terms, both of which may contain variables which refer to arbitrary terms. A rewrite system R defines a *rewrite relation* \rightarrow_R (or just \rightarrow) on terms, which is the smallest relation such that for all contexts $t[\diamond]$, rules $l \rightarrow r$ in R , and substitutions σ , $t[l\sigma] \rightarrow_R t[r\sigma]$. That is, if $t[l\sigma]$ has a subterm $l\sigma$ which is an instance of the left-hand side of a rule $l \rightarrow r$, then the R -redex $l\sigma$ in t is *contracted* by replacing that subterm with the corresponding instance $r\sigma$ of the right-hand side of the rule, thereby *rewriting* $t[l\sigma]$ to $t[r\sigma]$. A redex u is *innermost* for R if it is an R -redex but no proper subterms of u are R -redexes. An occurrence of a redex u in a term t is *outermost* for R if the occurrence of u is not a proper subterm of any other R -redexes of t . We often use R as an abbreviation for the binary relation \rightarrow_R . A rewrite rule $l \rightarrow r$ is *left-linear* if l is linear, and *right-linear* if r is. A rewrite system is *left-* (*right-*) *linear* if all its rules are. If function symbol f is not the outermost function symbol of the left-hand side of any rule in R , f is called a *constructor* for R .

As is commonplace, the reflexive-transitive closure of a binary relation \rightarrow is indicated by \rightarrow^* , its transitive closure is indicated by \rightarrow^+ , and its reflexive closure by $\rightarrow^=$. We write $s \leftarrow r$ if $r \rightarrow s$ and $r \leftrightarrow s$ if either $r \rightarrow s$ or $r \leftarrow s$. Composition of two relations is indicated by \circ . Thus $r \leftarrow^* \circ \rightarrow^* s$ if there is an element t such that $r \leftarrow^* t$ and $t \rightarrow^* s$.

An *equation* is a formula of the form $r = s$ where r and s are terms. An *equational system* is any set of equations. If E is a set of equations let E_{\rightarrow} be $\{r \rightarrow s : r = s \in E\}$. We write $r \leftrightarrow_E s$ if $s \rightarrow_E r$ or $r \rightarrow_E s$, where $u \rightarrow_E v$ is defined as $u \rightarrow_{E_{\rightarrow}} v$. We may write \rightarrow_E when we actually mean \leftrightarrow_E , viewing the equations as unordered pairs—as long as no confusion is likely to arise. If E is a set of equations and $>$ is an ordering on terms, we define the *ordered rewriting* relation of E as the pairs

of terms $\{(r, s) : r \leftrightarrow_E s \text{ and } r > s\}$. This relation may also be simply denoted by \rightarrow_E when the ordering in question is understood.

A *derivation* for a binary relation \rightarrow (for instance, a rewrite relation \rightarrow_R) is a sequence of the form $t_0 \rightarrow t_1 \rightarrow t_2 \cdots$. An element t is *reducible* (with respect to a given binary relation \rightarrow) if there is an element u such that $t \rightarrow u$; otherwise, t is *irreducible*. We say that u is an \rightarrow -*normal form* of t if $t \rightarrow^* u$ and u is irreducible via \rightarrow . We write $t \rightarrow^! u$ if $t \rightarrow^* u$ and u is a normal form. We also write R -reducible instead of \rightarrow_R -reducible and R -normal form instead of \rightarrow_R -normal form, et cetera. The normalizability relation $\rightarrow^!$ defines a partial function when normal forms are unique and a total function when it is always uniquely normalizing.

A binary relation \rightarrow is *terminating* (or *strongly normalizing*) if there are no infinite derivations $t_0 \rightarrow t_1 \rightarrow t_2 \cdots$. It is *terminating* for a set T of elements if there are no infinite derivations with $t_0 \in T$. A relation \rightarrow is *confluent* if there is an element v such that $s \rightarrow^* v$ and $t \rightarrow^* v$ whenever $u \rightarrow^* s$ and $u \rightarrow^* t$ for some elements s, t , and u . Confluence is equivalent to the *Church-Rosser* property that $s \rightarrow^* v$ and $t \rightarrow^* v$ for some v whenever $s \leftrightarrow^* t$. By extension, the terms "terminating" and "confluent" are also applied to rewrite systems R whose rewrite relation \rightarrow_R has those qualities. We say that a relation \rightarrow , or rewrite system R , is *convergent* (or *complete*) if it is terminating and confluent.² Convergent rewrite system are especially interesting, because all derivations lead to a unique normal form; such systems are used to decide equational theories (which describe *varieties*).

Suppose M is a first-order structure, with domain D , assigning meaning (semantics) to each individual constant, function symbol, and variable in the vocabulary. The meaning $[t]_M$ of a term t is an element of D and is defined inductively: $[x]_M = x^M$ for variable x with meaning x^M ; $[f(t_1, \dots, t_n)]_M = f^M([t_1]_M, \dots, [t_n]_M)$, if the meaning of f with arity n is f^M . If M_1 and M_2 are two structures and X is a set of variables, then $M_1 \equiv M_2 \pmod{X}$ if M_1 and M_2 agree on all function and constant symbols and on all variables not in X . If $s = t$ is an equation and all variables in s or t appear in a set X of variables, and M is a structure, then M *satisfies* $s = t$, written $M \models s = t$, if for all structures M' such that $M \equiv M' \pmod{X}$, $[s]_{M'} = [t]_{M'}$. One says that equation $s = t$ is *valid* in M , in this case. This corresponds to the intention that variables are implicitly universally quantified. A structure M is a *model* of E , written $M \models E$, if M satisfies each element of E . If E and E' are two equational systems, we write $E \models E'$ if all structures M that satisfy E also satisfy E' , that is, all models of E are also models of E' . In this case, if $E \models E'$, we call E' a *logical consequence* of E . The *equational theory* of E is the set of equations that are logical consequences of E .

Define R_\equiv to be the set of equations $l = r$ for each rule $l \rightarrow r$ in a system R . When R is convergent, $R_\equiv \models s = t$ (R_\equiv logically implies $s = t$) iff s and t have the same normal form with respect to R -rewriting. Thus we can use R for theorem proving in the equational theory R_\equiv . If R is not convergent, we may want to *complete* it, that is, find another system S such that their theories are the same

²The term *canonical* is sometimes used as a synonym for "convergent", but will not be so used here.

but S is convergent; then S may be used to decide the equational theory $R_{=}$ of R .

General rewriting logics have been devised [Goguen, Kirchner and Meseguer 1987, Cirstea and Kirchner 1999]. Such logics can be used to express many other logics in terms of the rewriting relation of a rewrite system.

3. Normal Forms and Validity

Rewriting methods have turned out to be among the more successful approaches to equational theorem proving. One main question of interest is to determine when $E \models s = t$ for various equation sets E and equations $s = t$. Intuitively, this means that the equation is necessarily true whenever all the equations in E are true. In order to answer this question, many proof systems have been developed, and recently many have been adapted for computer implementation. Rewriting can be used for this purpose:

3.1. THEOREM. *Suppose E is an equational system and \rightarrow is a binary relation. Then the equational theory E is decidable if the following conditions are satisfied:*

1. *If $s \rightarrow t$ then $E \models s = t$.*
2. *It is decidable whether a term is \rightarrow -reducible.*
3. *If r is \rightarrow -reducible, then one can compute a term s such that $r \rightarrow^+ s$.*
4. *The relation \rightarrow is terminating.*
5. *If r and s are irreducible for \rightarrow , then it is decidable whether $E \models r = s$.*

When the conditions are satisfied, one can reduce two given terms r and s to normal form using \rightarrow and test if they are E -equivalent. Presumably, testing if r and s are E -equivalent is easier when they are both irreducible. This theorem is used implicitly in the discussion of relativized (equational) rewriting in Section 7.

Rewrite systems are used in this way to check for validity. The simplest application of the above theorem is in the case when there is exactly one irreducible term in each E -equivalence class. Hence, one of the most essential properties a rewrite system can enjoy is *unique normalization*. In particular, convergent systems compute unique normal forms and serve as decision procedures for validity in the equational theory of E . For a convergent system R to determine provability in its underlying equational theory $R_{=}$, the test for reducibility must be recursive. Then, to decide if $s \leftrightarrow_R^* t$, one can check if the R -normal forms of s and t are identical.

A *rewrite ("valley") proof* of an equation $s = t$ for rewrite system R is a sequence $s_1, s_2, \dots, s_n, t_m, t_{m-1}, \dots, t_1$ where $s_1 \equiv s$ and $t_1 \equiv t$ and $s_n \equiv t_m$ and for all i , $1 \leq i < n$, $s_i \rightarrow_R s_{i+1}$ and for all i , $1 \leq i < m$, $t_i \rightarrow_R t_{i+1}$. This establishes the relation $s \rightarrow^* \circ \leftarrow^* t$ between s and t , indicating that the same term can be reached by rewriting s and t some number of times. The Church-Rosser property means that \leftrightarrow^* , as a relation, is equal to $\rightarrow^* \circ \leftarrow^*$; termination ensures that $\rightarrow^* \circ \leftarrow^* = \rightarrow^! \circ \leftarrow^!$; Church-Rosser implies that $\rightarrow^!$ defines a function; recursiveness of reducibility makes that function computable. Though the Church-Rosser property is undecidable, Knuth and Bendix [1970] devised an effective *superposition* test,

based on “critical overlaps”, to decide whether a terminating system is convergent. But termination itself is undecidable. Termination is further discussed in Section 4 and the Church-Rosser property in Section 5.

The (*ground*) *word problem* is to decide the truth of ground equations in equational theories (whose classes of models are called *varieties*); the *uniform word problem* is to decide validity of (universally quantified) equations. Of course, not all word problems can be solved by rewriting: some theories are not finitely based, and some finitely-based equational theories are undecidable.

Many rewrite-system decision procedures are known; perhaps the first such procedure for a word problem was Trevor Evans’ for loops [Evans 1951]; see Example 1.5. In their seminal paper, Knuth and Bendix [1970] (building on the work of Evans) demonstrated how failure of the superposition test suggests additional rules that can be used to help complete a nonconvergent system. Completion utilizes an ordering on terms to provide guidance in the generation of new rules and to direct the simplification of equations. *Ordered completion*, first suggested in [Brown 1975, Lankford 1975], and later developed further in [Hsiang and Rusinowitch 1987, Bachmair, Dershowitz and Plaisted 1989, Bachmair and Dershowitz 1994], is a powerful extension of Knuth’s method. Ordered completion permits unorientable equations to be used together with rewrite rules in the completion process.

Under weaker conditions, one can still use rewrite systems for theorem proving. A binary relation R is *normalizing* (or *weakly terminating*) if every term has an R -normal form, that is, for all r there is an s such that $r \rightarrow_R^! s$. A binary relation is *uniquely normalizing* if every term has exactly one normal form, though it need not necessarily be terminating. If a rewrite system R is finite and uniquely normalizing, one can test whether $R \models s = t$ by enumerating (in some breadth-first fashion) all derivations from s and t until their normal forms u and v are found, that is, $s \rightarrow_R^! u$ and $t \rightarrow_R^! v$. Then $R \models s = t$ iff u and v are identical. But this approach is often impractical, and we do not pursue it further.

In addition to its use for generating convergent systems to serve as decision procedures for the given axioms, ordered completion may also be used as an equational theorem prover. Classical forward reasoning systems work from the axioms, “expanding” the set of established formulæ by inferring new ones from old ones. Completion may be viewed as an inference engine that also “contracts” formulæ by constantly rewriting them, making forward reasoning practical. The potentially infinite set of rules and equations generated by completion are used to simplify the two sides of the equation in question. Even if a convergent system is not generated, it may be possible to generate enough rewrite rules to prove the theorem in question. Rules are used in the direction of the arrow only, while equations are used in whichever direction reduces the term it is applied to in the given ordering. An identity is proved when both sides reduce via rules and equations to the identical term. This is the subject of Section 8.

4. Termination Properties

One of the most important properties of a rewrite system, especially in the context of automated deduction, is termination.

The rules for the Chameleon Puzzle (1.3) are not terminating; to wit $RYY \rightarrow GGY \rightarrow GRR \rightarrow YYR$, which rearranges to RYY , from which point the same three steps may be repeated over and over again. On the other hand, each step in the Grecian Urn Puzzle (1.2) decreases the number of beans, except for the last two which shift beans around; so it always terminates if there is no infinite cycle of shifts. The Loop system also terminates, since it always shortens the length of the expression. Our Interpreter (1.6) is nonterminating.

The standard fundamental tool for termination proofs is the well-founded partial ordering. A partial ordering $>$ of a set W is *well-founded* if there are no infinite descending chains $x_0 > x_1 > x_2 > \dots$ of elements $x_i \in W$. A total well-founded ordering is a *well-ordering*. (As a consequence of Zorn's Lemma, every well-founded ordering is contained—as a set of pairs—in some well-ordering.) To make convenient use of well-founded orderings, we need several additional properties.

4.1. DEFINITION (*Monotonicity*). A binary relation \rightarrow on a set \mathcal{T} of terms satisfies the *monotonicity property* if $s \rightarrow t$ implies $u[s] \rightarrow u[t]$ for all terms s, t in \mathcal{T} and all contexts u over \mathcal{T} .

We will use the term “monotonic” in this sense only.

4.2. DEFINITION (*Stable Extension*). The *stable extension* of a binary relation \rightarrow on ground terms (over some vocabulary) to a relation on terms \mathcal{T} containing variables is defined so that $s \rightarrow t$ iff $s\sigma \rightarrow t\sigma$ for all $s, t \in \mathcal{T}$ and all ground substitutions σ .

The stable extension of a well-founded ordering is also well-founded. In practice, this extension is approximated in implementations by weaker orderings on free terms.

Stable extensions enjoy the following slightly more general property:

4.3. DEFINITION (*Full Invariance*). A binary relation \rightarrow on a set \mathcal{T} of free terms satisfies the *full invariance property* if for all terms $s, t \in \mathcal{T}$ and all substitutions σ over \mathcal{T} , $s \rightarrow t$ implies $s\sigma \rightarrow t\sigma$.

Length-based comparison of terms is a monotonic but not fully invariant ordering.

The rewrite step \rightarrow_R is by definition monotonic and fully invariant. So we are led to define:

4.4. DEFINITION (*Rewrite Relation*). A binary relation on terms is a *rewrite relation* if it is monotonic and fully invariant.

4.5. DEFINITION (*Reduction Ordering*). A well-founded partial ordering on terms is a *reduction ordering* if it is monotonic and fully invariant.

If $s > t$ under a reduction ordering $>$, then all variables in t must appear also in s . For example, if $f(x) > g(x, y)$, then by definition we also have $f(x) > g(x, f(x))$, and by monotonicity $g(x, f(x)) > g(x, g(x, f(x)))$, etc., giving an infinite descending sequence of terms. It follows that:

4.6. THEOREM (Lankford 1977). A rewrite system R is terminating if for some reduction ordering $>$, $l > r$ for all rules $l \rightarrow r \in R$ (symbolically: $R \subseteq >$).

The Loop system can be proved terminating using the fully-invariant *subterm ordering* $>_{\text{sub}}$ defined by $r >_{\text{sub}} s$ if s is a proper subterm of r .

In fact, whenever (finite or infinite) R is terminating, there is such a reduction ordering, viz. the derivability relation \rightarrow_R^+ . Thus, reduction orderings are (in theory) necessary and sufficient to prove termination of any terminating system R . But to be useful, reduction orderings for proving termination should be computable, or at least given to computable approximation, so that one can test whether $l > r$ for each rule.

If R is a finite rewrite system over a set of terms \mathcal{T} and t is a term in \mathcal{T} , define $R^\#(t)$ to be the length of the longest derivation beginning with t , and ∞ if there is an infinite derivation beginning with t . Now, R is terminating if for all t in \mathcal{T} , $R^\#(t)$ is finite. When $R^\#(t)$ is finite, the set of terms u such that $t \rightarrow_R^* u$ is finite, by König's Lemma.

4.7. THEOREM (Huet and Lankford 1978). The following problem is semi-decidable but not decidable: Given a finite rewrite system R and a term t , does R terminate for t ?

The construction is standard [Yasuhashi 1971].

In general, termination of even one rule systems is undecidable [Dauchet 1992]. For systems in which right-hand sides are ground, termination is decidable [Huet and Lankford 1978, Dershowitz 1981]. The concept of *fair termination* was defined in [Porat and Francez 1985]; the idea is that one only considers derivations in which no redex remains forever in the derivation without being contracted. Fair termination is decidable for ground systems [Tison 1989].

Since termination of all derivations initiated by a given term is undecidable, and termination for all terms is not even partially decidable, all one can hope for is practically useful termination tools.

Quasi-orderings are often more convenient than partial orderings for termination arguments. We will say that a quasi-ordering is well-founded, or is a reduction ordering, whenever its strict part is.

4.8. DEFINITION (*Sequence Ordering*). If \succ_i are quasi-orderings of sets of elements S_i , then the lexicographic extension \succ_{lex} of the \succ_i to arbitrary-length sequences in

$\cup_n(S_1 \times S_2 \times \cdots \times S_n)$ is defined as follows:³

$$\begin{array}{c} \frac{}{\langle \rangle \approx_{\text{lex}} \langle \rangle} \quad \frac{\langle s_1, \dots, s_m \rangle \lesssim_{\text{lex}} \langle t_1, \dots, t_n \rangle}{\langle s_1, \dots, s_m, s_{m+1} \rangle \succ_{\text{lex}} \langle t_1, \dots, t_n \rangle} \\[10pt] \frac{\langle s_1, \dots, s_n \rangle \approx_{\text{lex}} \langle t_1, \dots, t_n \rangle \quad s_{n+1} \approx_{n+1} t_{n+1}}{\langle s_1, \dots, s_{n+1} \rangle \approx_{\text{lex}} \langle t_1, \dots, t_{n+1} \rangle} \\[10pt] \frac{\langle s_1, \dots, s_m \rangle \approx_{\text{lex}} \langle t_1, \dots, t_m \rangle \quad s_{m+1} \succ_{m+1} t_{m+1}}{\langle s_1, \dots, s_{m+1} \rangle \succ_{\text{lex}} \langle t_1, \dots, t_{m+1}, \dots, t_n \rangle} \end{array}$$

where \lesssim_{lex} is the union of the equivalence relation \approx_{lex} and the partial ordering \succ_{lex} .

It is not hard to see that the relation \lesssim_{lex} is reflexive and transitive. When each of the element orderings is well-founded, the sequence ordering is also well-founded for bounded-length tuples (only).

Bags (a.k.a. *multisets*) are unordered collections of elements, in which multiplicity of elements matters, and for which we will use square brackets. Informally, a bag (e.g. $[2, 1, 1, 2, 3]$) is a set in which an element can occur more than once. Formally, a bag B is a function from some underlying domain to the nonnegative integers. Thus, $x \in B$ if $B(x) > 0$. A bag is finite if $\{x : B(x) > 0\}$ is finite.

4.9. DEFINITION (*Bag Ordering* [Dershowitz and Manna 1979]). The *bag (multi-set) extension* \lesssim_{bag} of a quasi-ordering \lesssim of elements is defined as follows:

$$\begin{array}{c} \frac{}{[\] \approx_{\text{bag}} [\]} \quad \frac{[s_1, \dots, s_n] \approx_{\text{bag}} [t_1, \dots, t_n] \quad s \approx t}{[s_1, \dots, s_n, s] \approx_{\text{bag}} [t_1, \dots, t_n, t]} \\[10pt] \frac{[s_1, \dots, s_m] \lesssim_{\text{bag}} [t_1, \dots, t_n] \quad s \succ u_1, \dots, u_k}{[s_1, \dots, s_m, s] \succ_{\text{bag}} [t_1, \dots, t_n, u_1, \dots, u_k]} \end{array}$$

where $s \succ u_1, \dots, u_k$ (for $k \geq 0$) means that s is greater than each of the u_i and \lesssim_{bag} is the union of the equivalence relation \approx_{bag} and the partial ordering \succ_{bag} .

The idea of this ordering is that a bag becomes smaller if an element is replaced by any number of smaller elements. Thus $[3, 4, 4] >_{\text{bag}} [2, 2, 2, 1, 2, 4, 4]$, since 3 has been replaced by a single 1 and four 2's. This operation can be repeated any number of times, still yielding a smaller bag; in fact, the relation $>_{\text{bag}}$ can be defined in this way as the smallest transitive relation having this property [Dershowitz 1987]. This relation can be computed reasonably quickly. It can be shown that the relation \lesssim_{bag} is transitive [Dershowitz and Manna 1979].

For a totally ordered underlying set, the elements of bags may be sorted in non-ascending order, then compared lexicographically; thus, the bag of ordinals

³An inference rule of the form $\frac{A_1 A_2 \dots A_n}{A}$ indicates that if A_1, A_2, \dots, A_n have been derived, then we may also derive A by using this rule. Such rules can also be used in a backward direction, meaning that if we are trying to derive A , or an instance of A , we attempt to derive all of A_1, A_2, \dots, A_n (or their instances).

$[\alpha_0, \dots, \alpha_n]$ is of order type $\sum \omega^{\alpha_i}$, summed naturally [Dershowitz and Manna 1979]. The bag ordering extends this effect to element orderings that are partial. For comparisons with alternative orderings on bags, see [Jouannaud and Lescanne 1982, Martin 1989].

4.10. THEOREM (Dershowitz and Manna 1979). *The bag (multiset) ordering is well-founded (for finite bags) iff the element ordering is.*

Term orderings used for proving termination of rewriting or completeness of rewrite-based theorem provers usually have the following properties:

4.11. DEFINITION (*Simplification Ordering* [Dershowitz 1982]). A monotonic quasi-ordering \succeq on terms is a (*quasi-*) *simplification ordering* if, for every function symbol f and index i ,

$$f(\dots, x_i, \dots) \succeq x_i$$

If f is varyadic, we also require

$$f(\dots, x_i, \dots) \succeq f(\dots, x_{i-1}, x_{i+1}, \dots)$$

A monotonic partial ordering $>$ on terms is a *strict simplification ordering* if for every function symbol f and index i ,

$$f(\dots, x_i, \dots) > x_i$$

If f is varyadic, we also require

$$f(\dots, x_i, \dots) > f(\dots, x_{i-1}, x_{i+1}, \dots)$$

For fixed-arity symbols, if a partial ordering is a rewrite relation and includes the subterm relation $>_{\text{sub}}$, then it is a fully-invariant simplification ordering. Simplification orderings are like the “divisibility order” of [Higman 1952], but apply to variable-arity function symbols. We will say that a simplification ordering is well-founded if its strict counterpart is.

4.12. DEFINITION (*Homeomorphic Embedding*). The *homeomorphic embedding* relation \geq_{emb} on a set \mathcal{T} of fixed-arity terms is the derivability relation of the rewrite rules

$$f(x_1, \dots, x_i, \dots, x_n) \rightarrow x_i$$

where x_1, \dots, x_n are distinct variables, for every f in the vocabulary and for each i . For variable-arity symbols f , we also have rules

$$f(\dots, x_i, \dots) \rightarrow f(\dots, x_{i-1}, x_{i+1}, \dots)$$

If $t \leq_{\text{emb}} s$, that is, if $s \rightarrow^* t$ in this system, we say that t is *embedded* in s . If t is embedded in s but they are not identical, then we write $t <_{\text{emb}} s$. Homeomorphic embedding is a monotonic quasi-ordering.

The crucial point is that simplification orderings \succeq contain the homeomorphic embedding relation \geq_{emb} . Strict simplification orderings $>$ contain the strict embedding $>_{\text{sub}}$.

4.13. THEOREM (Dershowitz 1982). *For terms over a finite set of function symbols and constants, any (quasi-) simplification ordering is well-founded.*

This follows directly from Kruskal's Tree Theorem:

4.14. THEOREM (Tree Theorem [Kruskal 1960]). *Any infinite sequence t_1, t_2, \dots of terms in a set \mathcal{T} with finitely many function symbols and constants contains two terms t_j and t_k ($j < k$) such that $t_j \leq_{\text{emb}} t_k$.*

Kruskal extended Higman's [1952] work to varyadic function symbols; for fixed-arity terms, this is "Higman's Lemma".⁴

Since fully-invariant strict simplification orderings (for finite vocabularies) are reduction orderings, they can be used to show termination, as per Theorem 4.6.

4.15. DEFINITION (*Self-Embedding*). A derivation $t_1 \rightarrow t_2 \rightarrow \dots \rightarrow t_j \rightarrow \dots \rightarrow t_k \rightarrow \dots$ of terms is *self-embedding* if $t_j \leq_{\text{emb}} t_k$ for some $j < k$. A rewrite system is *self-embedding* if it allows a self-embedding derivation. A rewrite system R is *self-embedding on t* if there is a self-embedding R -derivation initiated by t .

4.16. COROLLARY ([Dershowitz 1982]). *If a finite rewrite system is nonterminating, then it must be self-embedding.*

4.17. DEFINITION (*Simple Termination* [Ferreira and Zantema 1993]). A rewrite system R is *simply terminating* if there is a fully-invariant strict simplification ordering $>$ containing R .

We have seen (Theorem 4.13) that a rewrite system is terminating in this case. There are, however, terminating systems that are not simply terminating, such as the lone rule $f(f(x)) \rightarrow f(g(f(x)))$.

4.18. THEOREM (Dershowitz 1982). *Suppose R is a finite rewrite system and \succsim is a fully-invariant simplification ordering. If $R \subseteq \succsim$, then R is terminating.*

PROOF. Suppose $s_1 \rightarrow_R s_2 \rightarrow_R \dots \rightarrow_R s_n \rightarrow_R \dots$. Then $s_1 \succsim s_2 \succsim s_3 \dots$ because the quasi-ordering \succsim is monotonic. Let $B(s_i)$ be the bag of subterms of s_i . One can show that $B(s_1) \succ_{\text{bag}} B(s_2) \succ_{\text{bag}} B(s_3) \dots$, since some subterm t of s_i has been replaced by a strictly smaller rewritten term t' to obtain s_{i+1} , and all subterms of t' are smaller than, or equivalent to, s_i in the simplification ordering. Since R is finite, only finitely many function symbols appear in the derivation sequence. Since \succ is well-founded for terms over a finite vocabulary, \succ_{bag} is also. Therefore, the derivation must be finite and R terminates. \square

⁴Both Higman's Lemma and Kruskal's Theorem also apply to infinite vocabularies, with an underlying well-quasi-ordering of the symbols, extended to a homeomorphic embedding on terms. A beautiful, non-constructive proof may be found in [Nash-Williams 1963].

Instead of working directly with an ordering on terms, one can compare measures of terms in some well-founded ordering. For this purpose we have the following definition:

4.19. DEFINITION (Termination Function). A (quasi-) termination function τ is a function from a set of terms \mathcal{T} to a set W , supplied with a well-founded (quasi-) ordering \succeq that has the monotonicity property, such that for all terms in \mathcal{T} ,

$$\tau(f(s_1, \dots, s_n)) \succeq \tau(s_1), \dots, \tau(s_n)$$

For variable-arity symbols f , we also require

$$\tau(f(\dots, x_i, \dots)) \succeq \tau(f(\dots, x_{i-1}, x_{i+1}, \dots))$$

4.20. THEOREM. Suppose R is a rewrite system over a set \mathcal{T} of terms and τ is a termination function, then R is terminating if for all rules $r \rightarrow s$ in R and for all substitutions σ over \mathcal{T} ,

$$\tau(r\sigma) \succ \tau(s\sigma)$$

This is the method of [Manna and Ness 1970], adapted to quasi-orderings by analogy to Theorem 4.18.

Typically, termination functions $\tau : \mathcal{T} \rightarrow W$ are defined as a set of homomorphisms $\tau_f : W^n \rightarrow W$, one for each $f \in \mathcal{F}$ of arity n :

$$\tau(f(s_1, \dots, s_n)) = \tau_f(\tau(s_1), \dots, \tau(s_n))$$

Each variable x is mapped by τ to a distinct variable over W . When the termination function is defined by a convergent rewrite system, its necessary properties can be established using techniques similar to those of the next section; see [Bachmair and Dershowitz 1986, Bellegarde and Lescanne 1990].

As examples of termination functions, let W be the natural numbers and obtain $\tau(t)$ equal to the size (number of symbols) of t by defining $\tau_f(x_1, \dots, x_n) = 1 + x_1 + \dots + x_n$ for all symbols $f \in \mathcal{F} \cup \mathcal{C}$. This permits us to prove termination if all rules of R are of the form $r \rightarrow s$ where the size of all instances of r is larger than the size of corresponding instances of s , as in the Loop example. We can let W be the nonnegative reals and differentially weigh symbols by letting $\tau_f(x_1, \dots, x_n) = c_f + x_1 + \dots + x_n$, where c_f is a positive real number depending on f . Or we can obtain a quasi-termination function to the naturals by letting $\tau_f(x_1, \dots, x_n) = 1 + \max\{x_i\}$, which computes the depth of a term and shows termination when all rules are depth-reducing. Loops (Example 1.5) may be also be shown terminating by the depth measure.

More generally, one can devise termination functions by letting τ_f be an arbitrary multivariate polynomial in \bar{x} with integer coefficients, but then it is necessary to verify monotonicity for each such polynomial separately. Theorem proving techniques can be applied to this problem; implementations include [Ben Cherifa and Lescanne 1987, Steinbach 1994, Giesl 1995]. One must also show that

$\tau_f(x_1 \dots x_n) \geq 0$ if $x_i \geq 0$ for all i , $1 \leq i \leq n$. It is necessary to restrict W to the natural numbers so that the ordering $>$ on integers is well-founded. Of course, if all the coefficients are positive integers, these conditions hold. This technique has the advantage of flexibility, since different polynomials can be designed for different systems; however, there are some systems for which no such polynomial ordering will work. It is known that if the termination of R can be shown using such polynomials, then the length of the reduction sequences from a term t is at most doubly-exponential in the size of t [Hofbauer and Lautemann 1989]. This limits the kind of functions that can be computed by systems whose termination can be shown with these orderings. Also, it is not easy to devise appropriate polynomials for given rewrite systems.

Applying Theorem 4.13, we can extend polynomial orderings to the real numbers. Though the range of the polynomials is not well-ordered, termination is guaranteed by the theorem; see [Dershowitz 1982]. An interesting sidelight is that it is then possible to use a decision procedure for quantified inequalities involving polynomials over the reals, such as Tarski's original method [Tarski 1951] or the "reasonably practical" cylindrical algebraic decomposition algorithm [Arnon, Collins and McCallum 1984], to decide if polynomials of given degree exist that prove termination [Dershowitz 1979], not just to check that given polynomials over the integers show a decrease with each rewrite [Lankford 1975]. This can automate the method, but at a considerable cost, since the best upper bounds for the $\exists\forall$ fragment of reals (with addition, multiplication, and inequalities) are quite large (multiply-exponential in the number of variables) [Renegar 1992]. Some heuristics for obtaining such polynomials are given in [Steinbach 1994].

One can obtain evidence that a system R is terminating by demonstrating that $R^\#(t)$ is finite for "convincingly" many terms t , which is necessarily the case if R is terminating. (Recall that $R^\#(t)$ is the length of the longest derivation beginning with t .) Indeed, if R is terminating, then $s \rightarrow t$ implies $R^\#(s) > R^\#(t)$. However, $R^\#$ need not be monotonic. In addition, it cannot be defined as a homomorphism, since $R^\#(f(t_1, \dots, t_n))$ depends not only on $R^\#(t_i)$, but also on the structure of the t_i .

We can also secure evidence that a system R is not simply terminating, as follows:

4.21. THEOREM. *Given a finite rewrite system R and a term t , it is decidable whether R is self-embedding on t .*

PROOF. If there is a self-embedding of R , one can be found by enumerating derivations from t until a self-embedding is found. If no such self-embedding exists, then by Theorem 4.16 no derivation from t can be infinite, and thus there are only finitely many such derivations, by König's Lemma. These can be listed exhaustively, demonstrating that R is not self-embedding on t . In either case, it is decidable whether a given system R is self-embedding on t . \square

It is undecidable if there exists some term for which a given system is self-embedding [Plaisted 1985]. If R is self-embedding for some t , then R cannot be

simply terminating. On the other hand, if R is simply terminating, then this can often be demonstrated, since many computable simplification orderings are known (as we will see below). Combining these approaches, one may be able to determine automatically whether a system is simply terminating.

There are also techniques which can show nontermination of rewrite systems; for example, a rewrite system R is nonterminating if there is a term r , context u , and substitution σ such that $r \rightarrow_R^+ u[r\sigma]$. By using methods for proving termination together with this technique for establishing nontermination, one may be able to decide automatically for many naturally arising systems whether they are terminating.

The recursive path ordering [Dershowitz 1982] and its many variants have proved to be very useful for proving termination of rewrite system. These orderings can, for example, easily handle the distributive rule $x \times (y + z) \rightarrow x \times y + x \times z$. In addition, for many common systems, it is easy to find a recursive path ordering that suffices for termination proofs, in contrast to polynomial orderings. These orderings are defined in terms of a *precedence* on function symbols, which is a quasi-ordering on the symbols.

There are two main versions of the recursive path ordering, one based on bags of subterms (the multiset path ordering) and the other on sequences (the lexicographic path ordering). We begin with the sequence version:

4.22. DEFINITION (*Lexicographic Path Ordering* [Kamin and Lévy 1980]). The *lexicographic path ordering* \succ_{lpo} induced by a quasi-ordering \succeq on function symbols is defined as follows:

$$\frac{s_i \succeq_{\text{lpo}} t}{f(s_1, \dots, s_m) \succ_{\text{lpo}} t}$$

$$\frac{f \succ g \quad f(s_1, \dots, s_m) \succ_{\text{lpo}} t_1, \dots, t_n}{f(s_1, \dots, s_m) \succ_{\text{lpo}} g(t_1, \dots, t_n)} \quad \frac{f \approx g \quad \langle s_1, \dots, s_m \rangle \approx_{\text{lex}} \langle t_1, \dots, t_n \rangle}{f(s_1, \dots, s_m) \approx_{\text{lpo}} g(t_1, \dots, t_n)}$$

$$\frac{f \approx g \quad \langle s_1, \dots, s_m \rangle \succ_{\text{lex}} \langle t_1, \dots, t_n \rangle \quad f(s_1, \dots, s_m) \succ_{\text{lpo}} t_2, \dots, t_n}{f(s_1, \dots, s_m) \succ_{\text{lpo}} g(t_1, \dots, t_n)}$$

where \succ_{lpo} is the union of the equivalence relation \approx_{lpo} and the partial ordering \succ_{lpo} and \succ_{lex} and \approx_{lex} are the lexicographic extensions of \succ_{lpo} and \approx_{lpo} , respectively (see Definition 4.8).

The lexicographic path ordering is a simplification ordering for systems having fixed-arity function symbols [Kamin and Lévy 1980]. If the precedence \succeq is total, so is \succ_{lpo} .

This ordering has the useful property that $f(f(x, y), z) \succ_{\text{lpo}} f(x, f(y, z))$; informally, the reason is that—although the terms have the same size—the first subterm $f(x, y)$ of $f(f(x, y), z)$ is always larger than the first subterm x of $f(x, f(y, z))$. In this ordering, one can easily prove termination of Ackermann's function, for instance.

4.23. **EXAMPLE (Distributivity).** Suppose $\times \succ +$ in the precedence on function symbols. Then we can show that $x \times (y + z) \succ_{\text{lpo}} x \times y + x \times z$ (in the stable extension of the ordering), as follows:

$$\begin{array}{c}
 \frac{x \approx_{\text{lpo}} x \quad \frac{y \approx_{\text{lpo}} y}{y + z \succ_{\text{lpo}} y}}{\langle x, y + z \rangle \succ_{\text{lex}} \langle x, y \rangle} \quad \frac{\frac{y \approx_{\text{lpo}} y}{y + z \succ_{\text{lpo}} y}}{x \times (y + z) \succ_{\text{lpo}} y} \quad \frac{x \approx_{\text{lpo}} x \quad \frac{z \approx_{\text{lpo}} z}{y + z \succ_{\text{lpo}} z}}{\langle x, y + z \rangle \succ_{\text{lex}} \langle x, z \rangle} \quad \frac{\frac{z \approx_{\text{lpo}} z}{y + z \succ_{\text{lpo}} z}}{x \times (y + z) \succ_{\text{lpo}} z} \\
 \hline
 \frac{x \times (y + z) \succ_{\text{lpo}} x \times y \quad \times \succ + \quad x \times (y + z) \succ_{\text{lpo}} x \times z}{x \times (y + z) \succ_{\text{lpo}} x \times y + x \times z}
 \end{array}$$

4.24. **DEFINITION (Multiset Path Ordering [Dershowitz 1982]).** The multiset path ordering \succ_{mpo} , induced by a quasi-ordering \succsim on function symbols, is defined as follows:

$$\begin{array}{c}
 \frac{s_i \succsim_{\text{mpo}} t}{f(s_1, \dots, s_m) \succ_{\text{mpo}} t} \quad \frac{f \succ g \quad f(s_1, \dots, s_m) \succ_{\text{mpo}} t_1, \dots, t_n}{f(s_1, \dots, s_m) \succ_{\text{mpo}} g(t_1, \dots, t_n)} \\
 \frac{f \approx g \quad [s_1, \dots, s_m] \approx_{\text{bag}} [t_1, \dots, t_n]}{f(s_1, \dots, s_m) \approx_{\text{mpo}} g(t_1, \dots, t_n)} \quad \frac{f \approx g \quad [s_1, \dots, s_m] \succ_{\text{bag}} [t_1, \dots, t_n]}{f(s_1, \dots, s_m) \succ_{\text{mpo}} g(t_1, \dots, t_n)}
 \end{array}$$

where \succ_{mpo} is the union of the equivalence relation \approx_{mpo} and the partial ordering \succ_{mpo} and \succ_{bag} is the bag extension of \succ_{mpo} , as in Definition 4.9.

The multiset path ordering is a simplification ordering [Dershowitz 1982]. When the precedence is total, the ordering is total up to permutations of arguments.

Suppose we want to show that $f(g(a, b), d) \succ_{\text{mpo}} f(g(b, a), c)$ if $d \succ c$. (Recall that the precedence on function symbols is a quasi-ordering.) Using the above inference rules, we have that $g(a, b) \approx_{\text{mpo}} g(b, a)$ hence $[g(a, b), d] \succ_{\text{mul}} [g(b, a), c]$ so $f(g(a, b), d) \succ_{\text{mpo}} f(g(b, a), c)$.

4.25. **EXAMPLE (Permuted Distributivity).** Suppose $\times \succ +$ in the precedence. Then we can show that $x \times (y + z) \succ_{\text{mpo}} y \times x + z \times x$ (in the stable extension of the ordering), as follows: we have:

$$\begin{array}{c}
 \frac{x \approx_{\text{mpo}} x \quad \frac{y \approx_{\text{mpo}} y}{y + z \succ_{\text{mpo}} y}}{[x, y + z] \succ_{\text{bag}} [y, x]} \quad \frac{x \approx_{\text{mpo}} x \quad \frac{z \approx_{\text{mpo}} z}{y + z \succ_{\text{mpo}} z}}{[x, y + z] \succ_{\text{bag}} [z, x]} \\
 \hline
 \frac{x \times (y + z) \succ_{\text{mpo}} y \times x \quad x \times (y + z) \succ_{\text{mpo}} z \times x}{x \times (y + z) \succ_{\text{mpo}} y \times x + z \times x} \quad \times \succ +
 \end{array}$$

To prove that Insertion Sort (Example 1.4) terminates using \succ_{mpo} , note that five of the rules show a decrease for \succ_{mpo} by virtue of the subterm case of the definition. For the recursive rules of *max* and *min*, we can use the precedence $\text{max}, \text{min} \succ s$. For the base case of *insert*, we let *insert* \succ : in the precedence. For the recursive rule of *sort*, we let *sort* \succ *insert*. Finally, for the recursive rule of *insert*, we make *insert* \succ *max*, *min* in the precedence.

The multiset path ordering can be used to show that Hercules (Example 1.1) is invincible, considering Hydra as a term, nodes as a varyadic function symbol, and head as some constant symbol. We need to show that after each chop and regrowth, Hydra is smaller than before in \succ_{mpo} . Removing a head makes a branch smaller in \succ_{mpo} . Replacing a branch with any number of smaller branches is a decrease in the multiset path ordering. Thus any sequence of chops and regrowths terminates, which means Hercules wins as long as he keeps chopping, regardless of what Hydra does to retaliate.

An advantage of path orderings is that the precedence is often quite natural. For example, suppose, for some function f , $f(s)$ is defined in terms of $f(t_i)$ for t_i simpler than s (say, t_i are subterms of s). In other words, we have rules like $f(s) \rightarrow u[f(t_1), \dots, f(t_n)]$ where u is some multi-hole context containing smaller recursive calls $f(t_i)$. We want to show that this rule is decreasing in the multiset path ordering. Suppose u can be expressed as a composition of functions that have previously been defined. Then we can choose the precedence so that f is greater than any previously defined symbol. Since $s \succ t_i$ for all i , $f(s) \succ f(t_i)$ for all i . Also, $f(s) \succ g(f(t_1), \dots, f(t_n))$ if $f(s) \succ f(t_i)$ for all i . Using this rule repeatedly, we obtain that $f(s) \succ u[f(t_1), \dots, f(t_n)]$, so the rule $f(s) \rightarrow u[f(t_1), \dots, f(t_n)]$ is decreasing.

4.26. EXAMPLE (*Factorial*). As an example, consider the following straightforward rewrite system to compute the factorial function:

$$\begin{array}{llll} 0 + x & \rightarrow & x & \quad s(x) + y \rightarrow s(x + y) \\ 0 \times x & \rightarrow & 0 & \quad s(x) \times y \rightarrow y + (x \times y) \\ \text{fact}(0) & \rightarrow & s(0) & \quad \text{fact}(s(x)) \rightarrow s(x) \times \text{fact}(x) \end{array}$$

For this system, we choose the precedence $\text{fact} \succ \times \succ + \succ s \succ 0$. This is natural, since factorial is defined in terms of multiplication, multiplication in terms of addition, and addition in terms of the constructors. Using this precedence with the multiset path ordering, we have $l \succ r$ for all rules $l \rightarrow r$. This can be seen by considering the “dominant” subterm on both sides of each rule, by which we mean the subterm with the largest outermost function symbol. For the rule $\text{fact}(s(x)) \rightarrow s(x) \times \text{fact}(x)$, the dominant term on the left is $\text{fact}(s(x))$ and on the right is $\text{fact}(x)$. Since $s(x) \succ x$, $\text{fact}(s(x)) \succ \text{fact}(x)$ so the left-hand side is larger in the ordering. In general, there may not be a single dominant term, in which case the ordering has to be examined more carefully. One can show that $l \succ r$ if some dominant term in l is larger than all the dominant terms in r .

For any primitive recursive function, there is a rewrite system R computing it whose termination can be shown using a multiset path ordering [Plaisted 1978]. Primitive recursive definitions are encoded in the obvious way (as in Example 4.26) and the precedence is just the hierarchy of definitions. Moreover, Hofbauer [1992] showed conversely that if the termination of R can be shown using a multiset path ordering, then R computes a primitive recursive function (that is, the

function from a term to its R -normal form is primitive recursive) and Weiermann [1995] showed that lexicographic path orderings imply multiply recursive derivation lengths. Adding certain restrictions to the definition of the path orderings guarantees polynomial computations [Cichon and Marion 1999].

Total recursive functions can also be expressed as terminating rewrite systems. A function f , defined by minimization of a predicate t , may be encoded via rules

$$\begin{aligned} f(\dots x_i \dots) &\rightarrow \mu_t(0, t(0, \dots x_i \dots), \dots x_i \dots) \\ \mu_t(n, s(z), \dots x_i \dots) &\rightarrow n \\ \mu_t(n, 0, \dots x_i \dots) &\rightarrow \mu_t(s(n), t(s(n), \dots x_i \dots), \dots x_i \dots) \end{aligned}$$

where $t(n, \dots x_i \dots)$ is a predicate, with the added proviso that $t(k, \dots x_i \dots)$ implies $t(j, \dots x_i \dots)$ for all $j > k$. (A nonzero value for $t(n, \dots x_i \dots)$ is interpreted as **true** and a zero value is interpreted as **false**.) The normal form of a term $f(\dots a_i \dots)$ is the minimum n such that $t(n, \dots a_i \dots) \neq 0$. We assume that t can also be evaluated by a collection of rewrite rules.

The multiset and lexicographic path orderings can be directly combined using the notion of “status” [Lescanne 1990]. The idea is that for some function symbols f , when $f(s_1, \dots, s_m)$ and $f(t_1, \dots, t_n)$ are compared, the subterms are compared recursively using the bag ordering, while for other function symbols, subterms are compared using the lexicographic ordering (with the subterms arranged from left-to-right or right-to-left or in any fixed order). The name “recursive path ordering” will be used to refer to this status-based combination \succ_{rpo} .

A number of relationships between termination orderings and large ordinals have been found; this is only natural since any well-ordering corresponds to some ordinal. It is nice that the recursive path ordering (for total precedence) and other term orderings provide intuitive and useful descriptions of large ordinals. For some relevant discussions, see [Dershowitz 1987, Gallier 1991].

It is useful to be able to define termination orderings that are partial orderings and to be able to extend them piecemeal to more powerful orderings.⁵ Therefore, an important issue is incrementality, which means that a stronger precedence makes a stronger ordering: An ordering \succ_o based on a precedence \succ has the *incrementality property* if whenever a precedence \succ' extends \succ , the induced ordering \succ'_o extends \succ_o . The path orderings have this property, which allows one to successively extend the path ordering, as needed to orient more and more rules. This is one reason why the use of partial orderings is more flexible than the direct use of ordinal notations.

There are many other termination orderings that are similar to the above ones, such as the path of subterms ordering [Plaisted 1978] and the recursive decomposition ordering [Jouannaud, Lescanne and Reinig 1982]. These agree when the precedence is total [Rusinowitch 1987] and they all enjoy the incrementality property. This suggests the possibility of computing a “maximal” ordering for this class, which is as powerful as all of them combined.

⁵One ordering \succ' extends another \succ if $x \succ y$ implies $x \succ' y$, that is, if $\succ \subseteq \succ'$, with orderings viewed as sets of pairs.

Define the maximal multiset path ordering for a given precedence \succsim , as follows:

$$\succsim_{\text{mpo}}^{\text{sup}} = \bigcap_{\succsim' \succeq \succsim} \succsim'_{\text{mpo}}$$

Only total extensions \succsim' of the precedence are considered. See [Detlefs and Forgaard 1985]. Computing the maximal ordering in this way may be expensive, since there may be many total extensions. By using the inference rules defining the path ordering in a goal-directed manner, it is possible to construct a reasonably efficient decision procedure for term inequalities in this ordering. For example, suppose the precedence is $f > b$ and $a > g$. The terms $f(a)$ and $g(b)$ are incomparable in the multiset path ordering. Nevertheless, we can show that in any total extension \succeq' of the precedence, $f(a) >_{\text{mpo}}' g(b)$. If $f \geq a$ then $f > g$, and f is the maximal symbol of the two terms and $f(a) >_{\text{mpo}}' g(b)$. If $a \geq f$ then a is the maximal symbol of the two terms, and $f(a) >_{\text{mpo}}' g(b)$. Since one or the other must hold in any total extension of the precedence, we have $f(a) >_{\text{mpo}}^{\text{sup}} g(b)$.

A more efficient way of determining if $s >_{\text{mpo}}^{\text{sup}} t$ is to construct a constraint B involving the precedence \geq and function symbols such that $s \geq_{\text{mpo}} t$ iff the constraint holds. This can be derived systematically from the rules for the multiset path ordering. To decide whether $s >_{\text{mpo}}^{\text{sup}} t$, we need to show that all extensions of \geq to a total precedence \succeq' satisfy the constraint. This avoids repeated computations on the term structures. The precedence \geq can itself be defined by a conjunction of inequalities of the form $f_1 > g_1$, $f_2 \geq g_2$, etc. Call this conjunction of inequalities on function symbols C and let Q be the axioms of total quasi-orderings, namely, reflexivity, transitivity, and totality. We need to show that $C \wedge Q \Rightarrow B$. This is a straightforward theorem-proving problem, and can be approached by a number of methods that are reasonably efficient on small formulæ. In addition, one can use theorem proving to help compute the stable extension.

Determining if a precedence exists that makes two ground terms comparable in the multiset path ordering is NP-complete [Krishnamoorthy and Narendran 1985], but an inequality on ground terms can be decided in quadratic time, using a dynamic programming algorithm. The first-order theory of these orderings is undecidable [Comon and Treinen 1997]. The existential fragment, needed for completion (see Section 6), of the recursive path ordering—at least for total precedences—was shown decidable in [Comon 1990, Jouannaud and Okada 1991], and NP-complete in [Nieuwenhuis 1993, Narendran, Rusinowitch and Verma 1998].

Another important class of orderings, the *numeric path orderings*, uses homomorphic interpretations, perhaps in conjunction with a precedence. The *Knuth-Bendix ordering* is an example of such a hybrid ordering. For the Knuth-Bendix ordering with variables, an algorithm to decide inequalities was given in [Dick, Kalmus and Martin 1990]. In [Korovin and Voronkov 2001] a polynomial-time algorithm is given to decide whether there is a fully-invariant extension of a given Knuth-Bendix ordering that orients a given rewriting system. If such an ordering exists, the algorithm computes its parameters. The existential fragment is NP-complete [Korovin and Voronkov 2000].

Polynomial interpretations, followed by a lexicographic comparison of subterms, were suggested in [Lankford 1979]. Of course, one can use classes of interpretations other than polynomials, though it may be harder to decide inequalities. More advanced uses of semantics in path orderings have been defined; see [Kamin and Lévy 1980, Dershowitz and Hoot 1995, Zantema 1995, Genet and Gnaedig 1997, Borralleras, Ferreira and Rubio 2000].

The method of *dependency pairs* [Arts and Giesl 2000] limits the pairs that need to be shown decreasing by a reduction ordering to establish termination. Observe that if a system is nonterminating, then there must be an infinite derivation with at least one redex at the top of a term. Also if a system is nonterminating, then there's an infinite derivation in which all proper subterms of every redex initiate only finite derivations. Thus, to show impossibility of any infinite rewrite derivation it suffices to show the existence of some well-founded (not necessarily monotonic) order $>$ such that $l\sigma > s\sigma$ for nonvariable subterms s of the right side of a rule $l \rightarrow r$, and that $l\sigma > t$ for all t derivable from $s\sigma$ without any top-level rewrite. One way to establish that this termination condition holds is to show that $u \geq' v$ whenever u rewrites to v (using a quasi-simplification ordering), but that $l\sigma > s\sigma$ for each nonvariable subterm s of the right-hand side using another, related (non-monotonic) ordering for which $u \geq' v$ implies $f(\dots u \dots) \geq f(\dots v \dots)$, for all f . Moreover, one can weed out sterile, finite derivations by using data-flow techniques. For example, terms s that are headed by a constructor should be minimal in the ordering.

For right-linear systems one need only show that there are no infinite *forward closures* [Dershowitz 1981]; the same is true for systems with no “critical pairs” (defined in the next section) [Geupel 1989]; for left-linear systems, it is enough to consider *overlap closures* [Guttag, Kapur and Musser 1983, Geupel 1989].

As a comparative example, consider the first four rules of Example 4.26:

$$\begin{array}{ll} 0 + x & \rightarrow x & 0 \times x & \rightarrow 0 \\ s(x) + y & \rightarrow s(x + y) & s(x) \times y & \rightarrow y + (x \times y) \end{array}$$

We can use the following termination function $\tau_0 : \mathcal{T} \rightarrow \mathbb{N}$:

$$\begin{array}{ll} \tau_0(0) & = 1 & \tau_0(s(x)) & = \tau_0(x) + 2 \\ \tau_0(x + y) & = 2\tau_0(x) + \tau_0(y) & \tau_0(x \times y) & = \tau_0(y) \cdot 2^{\tau_0(x)} \end{array}$$

Or, to avoid exponentials, we could use the lexicographic combination $\langle \tau_1(t), \tau_2(t) \rangle$ of two simpler interpretations:

$$\begin{array}{ll} \tau_1(0) & = 1 & \tau_2(0) & = 0 \\ \tau_1(s(x)) & = \tau_1(x) + 2 & \tau_2(s(x)) & = \tau_2(x) + 2 \\ \tau_1(x + y) & = \tau_1(x) + \tau_1(y) & \tau_2(x + y) & = 2\tau_2(x) + \tau_2(y) \\ \tau_1(x \times y) & = \tau_1(x) \cdot \tau_1(y) & \tau_2(x \times y) & = 0 \end{array}$$

Rather than pairs of interpretations, one could—interchangeably—interpret every term as a pair of numbers, and define homomorphisms for each function symbol

that extracted the appropriate components from the interpretations of subterms and then put them together again [Zantema 1994].

For the dependency pair method, one can use the “natural” interpretation:

$$\begin{array}{llll} \tau'(0) & = & 0 & \tau'(s(x)) & = & \tau'(x) + 1 \\ \tau'(x + y) & = & \tau'(x) + \tau'(y) & \tau'(x \times y) & = & \tau'(x) \cdot \tau'(y) \end{array}$$

to show that $u \rightarrow t$ implies $u \geq' t$. Then for $>$ use the termination function:

$$\begin{array}{llll} \tau(0) & = & \langle 0, 0 \rangle & \tau(s(x)) & = & \langle 0, 0 \rangle \\ \tau(x + y) & = & \langle 1, \tau'(x) \rangle & \tau(x \times y) & = & \langle 2, \tau'(x) \rangle \end{array}$$

which expresses the fact that the recursion is on the first argument and that multiplication is defined in terms of addition. We have $s(x) + y > s(x + y), x + y$ and $s(x) \times y > y + (x \times y), x \times y$. The top two rules can be ignored.

Of course, either the multiset or the lexicographic path ordering can be used, simply with the precedence $\times > + > s$. If however the fourth rule were changed to read $s(x) \times y \rightarrow (y \times x) + y$, then only the multiset version would work.

5. Church-Rosser Properties

A key property a rewrite system can enjoy is confluence, because it reduces validity testing to rewriting. To begin with:

5.1. THEOREM (Birkhoff's Theorem [1935]). *For any equational system E and terms t and u , $E \models t = u$ iff $t \leftrightarrow_E^* u$.*

In other words, $t = u$ holds in all models of the identities E iff there is a finite sequence v_1, v_2, \dots, v_n of terms such that $t \equiv v_1$ and $u \equiv v_n$ and for each i , v_{i+1} is obtained from v_i by replacing a subterm r of v_i by a term s , where the equation $r = s$ or the equation $s = r$ is an instance of an equation in E . This gives an inefficient method for deriving logical consequences of sets of equations. Paramodulation improves on the naïve search for proofs; see [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook).

Let R be a rewrite system $\{r_1 \rightarrow s_1, r_2 \rightarrow s_2, \dots\}$ and recall that $R_=$ is the associated equational system $\{r_1 = s_1, r_2 = s_2, \dots\}$. We write $t =_R u$ iff $R_= \models t = u$, that is, the equation $t = u$ is a logical consequence of the associated equational system. The relation $=_R$ is thus the smallest congruence relation generated by R , in algebraic terms. The relation $=_R$ is defined semantically, and the relation \rightarrow^* is defined syntactically. We would like to find relationships between these two concepts to be able to compute properties of $=_R$ and to find complete restrictions of the inference rules suggested by Birkhoff's Theorem. As we will see, when R has certain properties, some of them decidable, then $t =_R u$ iff some normal form of t is identical to some normal form of u .

For any binary relation \rightarrow , we say that s *derives* t when $s \rightarrow^* t$ and that s and t are *convertible* when $s \leftrightarrow^* t$. If there is an element u mutually derivable from s and t ($s \rightarrow^* u$ and $t \rightarrow^* u$), we write $s \downarrow t$ and say that they are *joinable*, and that $s = t$ has a *rewrite proof*. We write $s \uparrow t$, and say that s and t are *meetable*, if there is an r such that $r \rightarrow^* s$ and $r \rightarrow^* t$.

5.2. DEFINITION (Church-Rosser). A binary relation is *Church-Rosser* if any two elements are joinable whenever they are convertible. Symbolically: $\leftrightarrow^* \subseteq \downarrow$.

5.3. DEFINITION (Confluence). A binary relation \rightarrow is *confluent* if any two elements are joinable whenever they are meetable. Symbolically: $\uparrow \subseteq \downarrow$.

The meaning of this is that diverging derivations can always be “brought together”.

5.4. THEOREM. *A binary relation has the Church-Rosser property iff it is confluent.*

PROOF. Let \uparrow^n be the n -fold composition of \uparrow and \uparrow^* its reflexive-transitive closure. We show that $\uparrow \subseteq \downarrow$ (confluence) implies $\uparrow^* \subseteq \downarrow$ (Church-Rosser, since $\leftrightarrow^* = \uparrow^*$). Trivially, $\uparrow^0 \subseteq \downarrow$. For $n > 0$, we have

$$\uparrow^n = \uparrow^{n-1} \circ \uparrow \subseteq \uparrow^{n-1} \circ \downarrow$$

by confluence. By definition (whether $n = 1$ or not),

$$\uparrow^{n-1} \circ \downarrow \subseteq \rightarrow^* \circ \uparrow^{n-1} \circ \leftarrow^*$$

and by induction

$$\rightarrow^* \circ \uparrow^{n-1} \circ \leftarrow^* \subseteq \rightarrow^* \circ \downarrow \circ \leftarrow^* = \downarrow$$

□

Since $s \leftrightarrow_R^* t$ iff $s =_R t$, this theorem connects the equational theory of R with rewriting. In order to decide if $s =_R t$ it is only necessary to see if s and t have a common normal form.

Often, we are only interested in confluence for ground (variable-free) terms.

5.5. DEFINITION (Ground Confluence). A rewrite system R is *ground confluent* if for all ground terms r , if $r \rightarrow_R^* s$ and $r \rightarrow_R^* t$ then $s \downarrow_R t$.

In other words, the rewrite relation, restricted to ground terms, is confluent.

Top-down and bottom-up tree automata (see [Thomas 1990]) execute a special form of ground rewriting and have been successfully used for proving decision properties in the ground case. Confluence of ground systems is decidable [Dauchet, Tison, Heuillard and Lescanne 1987, Oyamauchi 1987]. The unique normal form property [Verma, Rusinowitch and Lugiez 2001] is also polynomial for ground systems. But ground confluence of non-ground systems is undecidable [Kapur, Narendran and Otto 1990]. The latter property is useful for the so-called “inductionless induction” method invented by Musser [1980]. See [Comon 2001] (Chapter 14 of this Handbook).

5.6. DEFINITION (*Convergence*). Terminating confluent relations are called *convergent*.

By extension, a rewrite system is called *convergent* if its rewrite relation is. Many such systems are known.

Termination means that a rewriting process, applied to a term, will eventually stop, no matter how the rules are applied. If R is terminating, we can always find a normal form of a term by any rewrite sequence continued long enough. However there can be more than one normal form. But a convergent relation R defines unique normal forms, and it can be viewed as a function $R(x)$ from elements x to their normal forms. A convergent rewrite system gives a decision procedure for its equational theory, since for terms r and s , $r =_R s$ iff $r \leftrightarrow_R^* s$ (by Birkhoff's theorem) iff $r \downarrow s$ (by confluence) iff $R(r) = R(s)$ (by termination). The latter is a directed form of theorem proving for such an equational theory.

The straightforward encoding of primitive recursive functions as rewrite systems using the successor notation for natural numbers is convergent.

The next property we define is interesting because it permits a proof of confluence without assuming termination:

5.7. DEFINITION (*Strong Confluence*). A binary relation R is *strongly confluent* if for all r, s , and t , $r \rightarrow s$ and $r \rightarrow t$ imply that $s \leftrightarrow t$, or s and t are identical, or there is a term u such that $s \rightarrow u$ and $t \rightarrow u$.

5.8. THEOREM (Newman 1942). *Strongly confluent binary relations are confluent.*

The proof is by induction on the length of rewrite sequences. We invite the reader to construct the proof, which is straightforward.

Huet [1980] defines a slightly weaker version of "strong confluence" which also allows for $s \rightarrow^* t$ together with $t \rightarrow^* s$, and which still gives confluence.

Rewrite systems are typically not strongly confluent, while ordinary confluence, as stated, looks like a difficult property to demonstrate. However, we will see that when R is terminating and finite, confluence is decidable.

5.9. DEFINITION. Binary relations \rightarrow_R and \rightarrow_S (*sub-*) *commute* if $\leftarrow_R \circ \rightarrow_S \subseteq \rightarrow_S^= \circ \leftarrow_R^=$.

Recall that $\rightarrow^=$ is the reflexive closure of \rightarrow , allowing for at most one step.

5.10. LEMMA (Hindley-Rosen Lemma [Hindley 1964, Rosen 1973]). *If two strongly confluent relations commute, then their union is confluent.*

The following widely-applicable criterion for confluence extends the Hindley-Rosen method by dividing a relation \rightarrow into a family of subrelations \rightarrow_i . For any set K of indices i , we define $\rightarrow_K = \bigcup_{i \in K} \rightarrow_i$.

5.11. THEOREM (van Oostrom 1994). *Let K be a well-founded set of indices, divided into two not-necessarily disjoint sets R and S . Suppose the relation $\leftarrow_i \circ \rightarrow_j$*

is contained in the relation $\rightarrow_{I'}^* \circ \rightarrow_{J'}^{\bar{}} \circ \rightarrow_{K'}^* \circ \leftarrow_{K'}^* \circ \leftarrow_{I'}^{\bar{}} \circ \leftarrow_{J'}^*$, for all $i \in R$ and $j \in S$, where $I' = \{k \in S \mid k < i\}$, $J' = \{k \in R \mid k < j\}$, and $K' = I' \cup J'$. Then \rightarrow_R and \rightarrow_S commute.

The proof uses a bag ordering that ignores “noise” steps with indices that are smaller than subsequent steps.

5.12. DEFINITION (Local Confluence). A binary relation \rightarrow is *locally confluent* (weakly Church-Rosser) if for all terms r , s , and t , if $s \downarrow t$ whenever $r \rightarrow s$ and $r \rightarrow t$.

The following result connects local and global confluence:

5.13. LEMMA (Diamond Lemma [Newman 1942]). A terminating binary relation is Church-Rosser iff it is locally confluent.

PROOF. Clearly Church-Rosser implies local confluence, even without termination. Suppose $s \leftrightarrow^* t$ for a locally confluent and terminating relation \rightarrow . Then there is some sequence r_1, r_2, \dots, r_n of elements such that $s \equiv r_1$ and $t \equiv r_n$ and for all i , $r_i \leftrightarrow r_{i+1}$. For such a conversion, consider the bag $S = [r_1, \dots, r_n]$, ordered by the well-founded bag extension of the well-founded ordering \rightarrow^+ (the relation is terminating). If for no element r_i do we have $r_{i-1} \leftarrow r_i \rightarrow r_{i+1}$ then $s \downarrow t$ immediately. Suppose for some r_i we have $r_{i-1} \leftarrow r_i \rightarrow r_{i+1}$. Since the relation is locally confluent, $r_{i-1} \downarrow r_{i+1}$. Thus there is another conversion between s and t in which r_i is replaced by all the elements participating in the derivation of $r_{i-1} \downarrow r_{i+1}$. These elements are all smaller than r_i in the element ordering. Let T be the bag for this new conversion. Since r_i in S has been replaced by smaller elements, $S \succ_{\text{bag}} T$. By induction on the well-founded ordering \succ_{bag} , the derivation for T can be brought to the desired form, that is, $s \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \leftarrow t_2 \leftarrow t_1 \leftarrow t$, so $s \downarrow t$. \square

In order to show that a rewrite system is locally confluent, it is necessary to consider *critical pairs*, the computation of which requires most general unifiers. A substitution α is a *unifier* of two terms s and t if $s\alpha$ is identical to $t\alpha$. In this case we say that s and t are *unifiable*. We say that a substitution α is *as general as* a substitution β relative to a pair $s = t$ of terms if there is a substitution γ such that $s\alpha\gamma \equiv s\beta$ and $t\alpha\gamma \equiv t\beta$. A substitution α is a *most general unifier* of terms s and t , denoted $\text{mgu}(s, t)$, if α is a unifier of s and t and α is as general as any other unifier of s and t .⁶

5.14. DEFINITION (Critical Pair [Knuth and Bendix 1970]). If $s \rightarrow t$ and $l \rightarrow r$ are two (not necessarily distinct) rewrite rules (but with variables renamed so that they are distinct) and μ is a most general unifier of l and a nonvariable subterm s' of s , then the equation $s\mu[r\mu] = t\mu$, where $r\mu$ has replaced $s'\mu$ ($= l\mu$) in $s\mu$, is a

⁶If s and t are unifiable then most general unifiers exist, are essentially unique, and are easy to compute. Unification is the subject of Chapter 8 of this Handbook [Baader and Snyder 2001].

critical pair of those rules. We also require either that the two rules are distinct or that s' is a proper subterm of s to prohibit the trivial critical pair of a rule with itself. A critical pair $u_1 = u_2$ is *joinable* if u_1 and u_2 are joinable. A critical pair is an *overlay* if s' is s (and the two rules $s \rightarrow t$ and $l \rightarrow r$ are distinct). A rewrite system is *non-overlapping* if there are no critical pairs between its rules (disallowing the trivial one between a rule and itself).

The idea is that the term $s\mu[l\mu]$ can rewrite either to $t\mu$, applying the first rule at the top, or can rewrite to $s\mu[r\mu]$, applying the second rule to the redex $l\mu$. So if $u_1 = u_2$ is a critical pair, then $u_1 \leftarrow o \rightarrow u_2$ is a minimalist prototypical non-rewrite proof. A finite rewrite system can only have a finite number of critical pairs.

Insertion Sort has one trivially joinable critical pair, $0 = 0$, formed from the base cases of the *max* and *min* functions.

5.15. LEMMA (Critical Pair Lemma [Knuth and Bendix 1970, Huet 1980]). *A rewrite system is locally confluent iff all its critical pairs are joinable.*

5.16. EXAMPLE (*Fragment of Group Theory*). Each of the rules

$$\begin{array}{llll} 0 + x & \rightarrow & x & \quad \quad \quad x + 0 & \rightarrow & x \\ (-x) + x & \rightarrow & 0 & \quad \quad \quad x + (-x) & \rightarrow & 0 \\ -0 & \rightarrow & 0 & \quad \quad \quad -(-x) & \rightarrow & x \\ (-x) + (x + y) & \rightarrow & y & \quad \quad \quad x + ((-x) + y) & \rightarrow & y \end{array}$$

follows from some combination of the three axioms: $x + 0 \rightarrow x$, $0 + x \rightarrow x$, and $(-x) + (x + y) \rightarrow y$. This system has numerous critical pairs, all of which are joinable. For example, the rules $x + (-x) \rightarrow 0$ and $x + ((-x) + y) \rightarrow y$ form a critical pair $x + 0 = -(-x)$, both sides of which reduce, via other rules, to x .

PROOF. One direction is trivial. For the other, suppose u is a term and $u \rightarrow_R s$ and also $u \rightarrow_R s'$ for rewrite system R . There must be contexts c and c' , rules $l \rightarrow r$ and $l' \rightarrow r'$ of R and substitutions σ and σ' such that $u \equiv c[l\sigma] \equiv c'[l'\sigma']$, $s \equiv c[r\sigma]$, and $s' \equiv c'[r'\sigma']$. We need to show that $s \downarrow_R s'$.

If the redexes $l\sigma$ and $l'\sigma'$ are disjoint in u , then $u \equiv t[l\sigma, l'\sigma'] s \equiv t[r\sigma, l'\sigma']$ and $s' \equiv t[l\sigma, r'\sigma']$, for some context t . Then immediately $s \downarrow_R s'$, since $s, s' \rightarrow_R t[r\sigma, r'\sigma']$.

Another possibility is that one redex is “inside a variable” of another; that is, $u \equiv c[l\sigma]$, with σ instantiating some variable x to a term $t[l'\sigma']$ containing an instance of l' . In this case, we can view $l\sigma$ as $l\tau[l'\sigma']$, where τ is the same as σ , except that instead of $x \mapsto t[l'\sigma']$ it maps x to $t[\diamond]$, with that hole filled by $l'\sigma'$. So $u \equiv c[l\tau][l'\sigma']$, $c' \equiv c[l\tau]$, $s \equiv c[r\sigma] \equiv c[r\tau][l'\sigma', \dots, l'\sigma']$, with one occurrence of $l'\sigma'$ filling each hole left by τ for an x that occurs in r , and $s' \equiv c'[r'\sigma'] \equiv c[l\tau][r'\sigma'] \rightarrow_R c[r\tau][r'\sigma', \dots, r'\sigma']$, while $s \equiv c[r\tau][l'\sigma', \dots, l'\sigma'] \rightarrow_R^* c[r\tau][r'\sigma', \dots, r'\sigma']$ by successive rewrites. Thus $s \downarrow_R s'$.

The only other possibility is that the redexes overlap in a nonvariable subterm. Then $u \equiv c[l\sigma]$, with $l \equiv d[t]$, and $t\sigma \equiv l'\sigma'$. In this case, the equation $s = s'$ must

contain an instance of a critical pair of R , that is, $s = s' \equiv c[r\sigma] = c[d\sigma[r'\sigma']] \equiv c[p\tau] = c[q\tau]$, for some critical pair $p = q$ or $q = p$ (which is $r\mu = d\mu[r'\mu]$ for most general unifier μ of l' and t), and substitution τ (such that $\mu\tau = \sigma$). Since all critical pairs are joinable, $p\tau \downarrow_R q\tau$ and, hence, $s \downarrow_R s'$. \square

5.17. COROLLARY. *Suppose R is a terminating rewrite system. Then R is convergent iff for all critical pairs $s = t$ of R , arbitrarily computed normal forms s' of s and t' of t are identical.*

PROOF. The system R is locally confluent by the Critical Pair Lemma 5.15. Since R is terminating, it is confluent by the Diamond Lemma 5.13. \square

The implication of this result is that if R is a (finite) terminating rewrite system, then its local confluence (and confluence) is decidable, since given a finite R , there are only finitely many critical pairs, and they can be computed (in polynomial time). However, often we are interested in systems that are not known to terminate and other methods are needed to establish confluence. Such methods are interesting, even though it is not possible to decide their equational theory by rewriting, because confluence still enables one to prove validity of equations between those terms having normal forms.

5.18. DEFINITION (*Encompassment* [Huet 1981]). A term s encompasses a term t if a subterm of s is an instance of t . We write $s \triangleright t$ if s encompasses t , but not vice-versa.

5.19. DEFINITION (*Reduced System* [Butler and Lankford 1980]). A system R is *reduced* (interreduced) if, for each rule $l \rightarrow r$ in R , the right-hand side r is irreducible (unrewritable) and if $l \triangleright l'$, then l' is irreducible (proper subterms of l are in normal form, as is any term more general than l).

The group theory fragment (Example 5.16) is reduced, but the Interpreter (1.6) isn't.

5.20. DEFINITION (*Reduced Convergent System*). A rewrite system is *reduced convergent* if it is confluent, terminating, and reduced.

It turns out that for a given reduction ordering $>$ and equational theory E , there is a unique reduced convergent system.

5.21. THEOREM (Uniqueness [Butler and Lankford 1980, Métivier 1983]). *Two reduced convergent (not necessarily finite) rewrite systems with the same equational theory (that is, their convertibility relations are the same) are identical (up to renaming variants), if their union is terminating.*

See [Dershowitz, Marcus and Tarlecki 1988] for details.

A rewrite system is locally confluent if (but not only if) no left-hand side unifies with a nonvariable subterm (except itself) of any left-hand side, taking into account

that variables appearing in two rules (or in two instances of the same rule) are always treated as distinct. To get confluence for nonterminating systems we impose an additional requirement of left-linearity:

5.22. DEFINITION (*Orthogonality*). A left-linear rewrite system is (*weakly*) *orthogonal* if all critical pairs are trivial (both sides identical).

In particular, all critical pairs are trivial if all of the following hold:

1. no left-hand side is just a variable;
2. there are no variables on the right side of a rule that do not appear also on the left;
3. no left-hand side unifies with a (not necessarily proper) non-variable subterm of another left side (renamed apart);
4. no left-hand side unifies with a proper non-variable subterm of a renamed variant of itself.

Orthogonality is usually defined in this particular sense, but the weaker notion suffices for most purposes. (Conditions 1 and 2 are often included in the very definition of rewrite system, but that is ill-advised.)

The importance of orthogonal systems stems from the following:

5.23. THEOREM (Rosen 1973). *Every orthogonal system is confluent.*

In particular, our erstwhile interpreter (1.6) is confluent.

PROOF. The idea is to show that a parallel rewriting relation $\rightarrow_R^{\parallel}$ associated with R is strongly confluent (since the system may be assumed nonterminating). Parallel rewriting is rewriting at one or more disjoint redexes at the same time. We need to consider parallel rewriting because if s rewrites to t and s rewrites to t' , at positions that are not disjoint, then a subterm of s may appear many times in t or t' , and all of these occurrences may have to be rewritten in parallel to obtain a v to which both t and t' rewrite in one (parallel) step. For example, if $s \equiv f(a)$ and R is $\{a \rightarrow b, f(x) \rightarrow g(f(x), f(x))\}$ then $f(a)$ rewrites to both $f(b)$ and $g(f(a), f(a))$. Both of these terms parallel-rewrite to $g(f(b), f(b))$, by replacing both occurrences of a in $g(f(a), f(a))$. Similar techniques are used to show the confluence of lambda calculus and combinatory logic, which do not terminate. \square

The following well-known example [Klop 1980] shows up the essentiality of left-linearity:

$$\begin{aligned} d(x, x) &\rightarrow e \\ c(x) &\rightarrow d(x, c(x)) \\ a &\rightarrow c(a) \end{aligned}$$

The system is non-overlapping because the outermost function symbols of each left-hand side are distinct, and there are no nonvariable proper subterms. Note that $c(a) \rightarrow^* e$ and $c(a) \rightarrow^* c(e)$, but we do not have $e \downarrow c(e)$.

Even though non-left-linearity removes the confluence guarantee, Chew [1981] suggested that some limited confluence properties still apply to some non-overlapping non-left-linear systems, in terms of the uniqueness of normal forms. For example, his methods imply that the above system has unique normal forms, despite its lack of confluence.

5.24. **EXAMPLE** (*Combinatory Logic*). Combinatory Logic is a prime example of a (nonterminating) orthogonal system:

$$\begin{aligned} I \circ x &\rightarrow x \\ (K \circ x) \circ y &\rightarrow x \\ ((S \circ x) \circ y) \circ z &\rightarrow (x \circ z) \circ (y \circ z) \end{aligned}$$

The combinators K and S were dubbed “kestrel” and “starling” by Smullyan [Smullyan 1968]; I is the identity combinator; \circ is composition. This system can be used to implement any recursive function.

5.25. **EXAMPLE** (*Cartesian Closed Categories*). The following non-orthogonal, non-confluent system [Huet 1985] is used in the compilation of functional languages (juxtaposition and pairing $\langle \diamond, \diamond \rangle$ are binary operators):

$$\begin{array}{ll} Ix &\rightarrow x & (xy)z &\rightarrow x(yz) \\ xI &\rightarrow x & \langle x, y \rangle z &\rightarrow \langle xz, yz \rangle \\ F\langle x, y \rangle &\rightarrow x & E\langle Cx, y \rangle &\rightarrow x\langle I, y \rangle \\ S\langle x, y \rangle &\rightarrow y & (Cx)y &\rightarrow C(x\langle yF, S \rangle) \end{array}$$

The combinators E and C stand for “evaluation” and “Currying”, respectively; I is the identity morphism; F and S project the components of pairs.

There are redex-choosing strategies for orthogonal systems that compute the unique (but not necessarily existent) normal form of a term; see Section 10. There are also some results about termination of orthogonal systems; for example, it is known that an orthogonal system is weakly innermost normalizing (every term has a normal form obtainable by some innermost rewriting sequence) iff it is terminating [Gramlich 1995].

Since the factorial example (4.26) is orthogonal, innermost termination implies (strong) termination. This allows one to prove termination by a straightforward natural interpretation and ordinary induction on the value of the first argument.

For orthogonal systems that are *non-erasing* (the left and right-hand sides of rules have the same variables appearing), a system is weakly normalizing (every term has a normal form) iff it is terminating [O’Donnell 1977], as is the case for the λI calculus. Left-linearity is not actually required for these results [Dershowitz and Hoot 1995, Gramlich 1995].

See [Raoult and Vuillemin 1980, Naoi and Inagaki 1989] for studies of denotational semantics and confluent systems. Though most research in term rewriting

has concentrated on confluent systems, there is some interest in systems that model nondeterminism [Kaplan 1988]. For such systems, we may introduce a nondeterministic choice operator with rules $[x|y] \rightarrow x$ and $[x|y] \rightarrow y$.

6. Completion

The impractical, "British Museum" approach to constructing a convergent system for a given equational theory E would be to generate all possible sets of equational consequences of E , orient as many as possible according to some given reduction ordering, and check each subset of the rules for completeness (all E follows) and confluence (all critical pairs between the resultant rules resolve). As a practical matter, a method, called *completion*, is used, which converts unresolved critical pairs into rewrite rules. Completion uses a reduction ordering to orient equations, rewriting to simplify rules and equations, and the encompassment ordering to determine which of two rules is more general, and hence preferred. Knuth and Bendix [1970] used their completion program (written in Fortran) to construct convergent systems for free groups, loops (Example 1.5 shown earlier was found manually by Evans [1951]), left and right groups, and central groupoids.

In general, inference systems have rules for combining theorems in a database of proved theorems to obtain new theorems which can be added to the database. The generic version of that process is

$$\text{Expand: } \frac{E}{E, C} \quad \text{if } E \models C$$

meaning that E is a collection of theorems that have been proved and are in the database and C is one or more consequences of E that can be added to the database.

Completion procedures, which are inference systems, incorporate another schema for deleting redundant entries in the database, ones that cannot contribute to minimal proofs. For that purpose, one employs a *proof ordering* \gg in conjunction with the inference rule:

$$\text{Contract: } \frac{E, C}{E} \quad \begin{array}{l} \text{if for every proof } p \text{ of } E \cup C \vdash e, \\ \text{there is a proof } q \text{ of } E \vdash e, \text{ such} \\ \text{that } p \gg q \end{array}$$

For such a theorem-proving method to be complete, every non-minimal proof must contain a subproof that can be reduced with the help of consequences that will eventually be added by the expansion rule.

To get the flavor of the general approach, consider first how rewriting can be used to transform a (finite) set E of ground equations into a simple decision procedure for E [Lankford 1975]. This transformation is expressed as a set of rules that use a total strict simplification ordering to *replace* equations with simpler ones, without changing the theory:

$$\frac{u = u, E}{E} \quad \frac{e[l], l = r, E}{e[r], l = r, E} \quad \text{if } l > r$$

The left rule deletes trivial equations and is the simplest contraction rule. The other simplifies an existing equation e using another equation, it being understood that e is distinct from $l = r$. This simplification is a combination of an expansion step that adds $e[r]$ and a contraction that removes $e[l]$.

In a completion procedure, these two rules are applied repeatedly as long as possible, assuming totality of the simplification ordering. Regardless of the order in which things are done, this inference process terminates. Done right, it completes a ground system E in time proportional to $n \log n$, where n is the number of symbols in E [Snyder 1989]. This process is also known as *congruence closure* (see, for example, [Nelson and Oppen 1980]). The final system of equations, call it R , reduces any term t to normal form in no more steps than symbols in t . An equation $s = t$ holds in E iff R reduces $s = t$ to a trivial equation of the form $u = u$.

For example, if terms are compared by length, then these rules can have the following effect:

$$\begin{array}{lcl} sss0 = 0, ss0 = 0, ssss0 = s0 & \vdash & sss0 = 0, ss0 = 0, ss0 = s0 \\ & \vdash & ss0 = 0, ss0 = s0 \\ & \vdash & ss0 = 0, 0 = s0 \\ & \vdash & s0 = 0 \end{array}$$

We are using $E \vdash E'$ to indicate one application of an inference rule. The result is the one-rule rewrite system, $s0 \rightarrow 0$, which reduces all terms s^i0 to normal form 0.

Turning to the general case, when the axioms in E have variables, the question is how one constructs an equivalent convergent system R , which can be used to check validity by the same method, viz. reducing to normal form. In the version of completion we present, rules and equations are used for simplification. A term $u[l\sigma]$ containing an instance of l of an equation $l = r$ or $r = l$ may be rewritten, via *ordered rewriting*, to $u[r\sigma]$ whenever $u[l\sigma] > u[r\sigma]$ in a given reduction ordering $>$. If reducibility is recursive, a set of equations E for which ordered rewriting \rightarrow_E is confluent computes unique normal forms. So, to prove an identity $s = t$, one *Skolemizes* its variables (treating its variables as constants), and uses the evolving \rightarrow_E^1 to reduce both its sides. A trivial equation $s' = t'$ is obtained eventually iff $E \models s = t$.

6.1. EXAMPLE (Groupoid). Consider the following system for a subvariety of entropic groupoids [Hsiang and Rusinowitch 1987]:

$$\begin{array}{ll} (xy)x \rightarrow x & x(yz) \rightarrow xz \\ (xy)z \leftrightarrow (xw)z & ((xy)w)z \rightarrow xz \end{array}$$

We are using the symbol \rightarrow for equations such that all instances of the left-hand side are greater than corresponding instances of the right-hand side, and \leftrightarrow , when the direction of application depends on the instance. Thus, the double-headed rule is used to rewrite any product of the form $(xy)z$ to the same term with the occurrence

of y replaced by a sufficiently small term in the ordering $>$, as though it were

$$y > w \mid (xy)z \rightarrow (xw)z$$

(In this example, it matters little which reduction ordering is used.) To prove validity of the identity $(xy)(wz) = (xw)(yz)$, both sides are rewritten. Applying the upper-right rule to both sides, we get $(xy)z = (xw)z$. Suppose a is smaller than any other term. Then the identity reduces to the trivial equation $(xa)z = (xa)z$.

By adding to the vocabulary a new constant a smaller than any term in the ordering $>$, one can reformulate the system as

$$\begin{array}{lll} (xa)x & \rightarrow & x \\ y \neq a \mid (xy)z & \rightarrow & (xa)z \end{array} \quad \begin{array}{ll} x(yz) & \rightarrow & xz \\ ((xa)a)z & \rightarrow & xz \end{array}$$

The condition $y \neq a$ insures that $(xy)z > (xa)z$. There is no need to consider other instances of the original rules, since terms to which they would apply can first be rewritten by the conditional rule.

Ordered rewriting always terminates, since the rewrite relation is contained in a reduction ordering. For confluence, we require a broader notion of critical pair than 5.14:

6.2. DEFINITION (Ordered Critical Pair [Lankford 1975]). Given a reduction ordering $>$, if $s = t$ (or $t = s$) and $l = r$ (or $r = l$) are two (not necessarily distinct) equations (but with variables made distinct) and μ is a most general unifier of l and a nonvariable subterm s' of s , and for some substitution σ , we have $s\mu\sigma \not\leq t\mu\sigma$ and $s\mu\sigma \not\leq s\mu[r\mu]\sigma$ in the ordering, then the equation $s\mu[r\mu] = t\mu$ (with $r\mu$ replacing $s'\mu$ in $s\mu$) is a (ordered) critical pair.

Let $\mathbf{cp}(E)$ denote the set of all (ordered) critical pairs between equations $l = r$ in a set E . When $l > r$, then by definition its instances are also ordered thus. So overlapping r on either side of another equation does not contribute a critical pair to $\mathbf{cp}(E)$. Also, if R is a set of rewrite rules, then $\mathbf{cp}(E \cup R) = \mathbf{cp}(E \cup R_{=})$, permitting critical pairs between rules and equations.

6.3. THEOREM (Ordered Critical Pair [Lankford 1975]). *Let the ordered rewrite relation \rightarrow_E be defined by an equational system E and reduction ordering $>$. If all ground instances $l\sigma = r\sigma$ of critical pairs $l = r \in \mathbf{cp}(E)$ have ordered rewriting proofs, $l\sigma \downarrow_E r\sigma$, then \rightarrow_E is ground confluent.*

PROOF. As in Lemma 5.15, for any peak $s \leftarrow_E \circ \rightarrow_E t$ between ground terms s, t , there either exists a rewrite proof $s \downarrow_E t$ or a critical-pair proof $s \leftrightarrow_{\mathbf{cp}(E)} t$. In the latter event, $s \equiv u[l\sigma]$ and $t \equiv u[r\sigma]$ for some critical pair $l = r$, and by assumption $u[l\sigma] \downarrow_E u[r\sigma]$. So the relation \rightarrow_E is locally confluent, and by the Diamond Lemma the rewrite relation \rightarrow_E is confluent for ground terms. \square

Though there are only a finite number of critical pairs for finitely many equations, the question is how, in general, one can check whether all instances have rewrite proofs. In the previous example (6.1), the critical pair $x = (xw)x$ (one of several between the first two rules) always reduces by another application of the first rule to $x = x$. On the other hand, the pair $((xx')y)z = ((xx)w)z$ can be rewritten to $((xx')w)z = ((xx')w)z$ by the third rule, but only because we must have $x > x'$ and $y > w$ for the critical pair to arise in the first place. In this case, it is actually no problem at all, since the third rule can be used instead to show that both sides have normal form xz .

In general, confluence of ordered rewriting is decidable for the wide class of precedence-based path orderings described in Section 4 [Comon, Narendran, Nieuwenhuis and Rusinowitch 1998].

Completion is used to construct convergent systems for a given set of equational axioms. The procedure maintains a set E of equations and a set R of rules, oriented according to a given reduction ordering $>$. Equations are only used to rewrite when they cause a decrease under $>$. These sets are manipulated by the following inference rules:

Deduce: Add a critical pair formed from left-sides of rules in R and both sides of equations in E :

$$\frac{E, R}{E, R, s = t} \quad \text{if } s = t \in \mathbf{cp}(E \cup R)$$

Orient: Orient an equation $l = r$ (or $r = l$) for which $l > r$:

$$\frac{E, R, l = r}{E, R, l \rightarrow r} \quad \text{if } l > r$$

Delete: Remove an equation whose sides are identical:

$$\frac{E, R, r = r}{E, R}$$

Simplify: Use a rule to rewrite either side of an equation (all equations are treated as unordered pairs) or use either a rule or equation to rewrite the right side of a rule (all rules are treated as ordered pairs):

$$\frac{E, R, l = r}{E, R, l = r'} \quad \text{if } r \rightarrow_R r' \qquad \frac{E, R, l \rightarrow r}{E, R, l \rightarrow r'} \quad \text{if } r \rightarrow_{E \cup R} r'$$

Collapse: Use equations or rules to rewrite a less general term:

$$\frac{E, R, s[l\sigma] = t}{E, R, s[r\sigma] = t} \quad \text{if } l = r \in E, l \text{ not a variant of } s$$

$$\frac{E, R, s[l\sigma] \rightarrow t}{E, R, s[r\sigma] \rightarrow t} \quad \text{if } l \rightarrow r \in R \text{ or } l = r \in E, l \text{ not a variant of } s$$

We begin with a finite set E of equations and no rules in R . The set of equations is expanded by deduction of critical pairs (the rule **deduce**). **Simplify**, in essence, first expands the set of rules to include $l \rightarrow r'$, since $l = r = r'$ and $l > r > r'$, and then contracts the rules by deleting $l \rightarrow r$ which, given $l \rightarrow r'$ and the fact that $r \rightarrow r'$, is redundant.

There is room for flexibility in how the inference rules are applied. One simple version of completion mixes the above inference steps according to the following strategy:

Completion = $((\text{Simplify} + \text{Delete})^*; (\text{Orient}; \text{Collapse})^*)^*; \text{Deduce})^*$

In words: Simplify and delete equations as much as possible before orienting. Then use the newly oriented equation to collapse left-hand sides of all nonreduced existing rules; then, go back and simplify over again. When there are no equations left to orient, generate one new critical pair, and repeat the whole process.

Different versions of completion differ in which equations they orient first and in how they keep track of critical pairs that still need to be deduced.

For guaranteed success of completion, the ordering should be total on ground terms:

6.4. DEFINITION (*Complete Simplification Ordering [Hsiang and Rusinowitch 1987]*).

A reduction ordering $>$ of a set of terms \mathcal{T} is called a *complete simplification ordering* if it totally orders all the ground terms in \mathcal{T} . A reduction ordering is *completable* if it can be extended to a complete simplification ordering.

Completable simplification orderings of necessity have the *subterm property*, which states that terms are larger than their proper subterms. Examples include the empty ordering, the lexicographic path ordering with a partial precedence, and the numeric path ordering. Orderings, such as the recursive path ordering, enjoying the incremental property can be gradually extended to cover new rules as they are generated. This is why precedences that are partial orderings are of importance even though total precedences yield more powerful termination orderings.

An implementation of completion is *fair* if it does not altogether avoid processing any relevant, nonredundant critical pair. Running completion with a fair strategy can have one of three outcomes: It might converge on a finite system of only rules that is a decision procedure for the initial set of equations; it might reach a point in which all (ordered) critical pairs have been considered and all have rewrite proofs—using rules *and* equations; or it might loop and generate an infinite number of rules and equations. The uniqueness of reduced convergent systems (Theorem 5.21) implies that the choice of ordering determines the final result of completion.

6.5. DEFINITION (*Fairness [Bachmair and Dershowitz 1994]*). An inference sequence $E_0 \vdash E_1 \vdash \dots$ is *fair* with respect to completion if

$$\text{cp}(E_\infty) \subseteq E_0 \cup E_1 \cup \dots$$

where $E_\infty = \liminf_j E_j (= \cup_{i \geq 0} \cap_{j \geq i} E_j)$ is the set of *persisting* equations.

The following can be shown by induction with respect to a suitable proof ordering \gg :

6.6. THEOREM (Completeness of Completion [Bachmair and Dershowitz 1994]). *For any fair completion strategy and complete simplification ordering, ordered rewriting with (finite or infinite) E_∞ is convergent.*

This means that eventually both sides of any identity in the theory of the initial set of equations E_0 will become joinable. Thus, fair completion is a complete equational theorem prover.

6.7. EXAMPLE (Abelian Groups I). Completion, given

$$\begin{array}{ll} x \cdot 1 = x & x \cdot y = y \cdot x \\ x \cdot (y \cdot z) = (x \cdot y) \cdot z & x \cdot x^- = 1 \end{array}$$

and a lexicographic path ordering (in which $- > \cdot > 1$ and left arguments are looked at first) will generate the following set of rules which constitute a decision procedure for free Abelian (commutative) groups under ordered rewriting:

$$\begin{array}{ll} 1^- \rightarrow 1 & x \cdot y \leftrightarrow y \cdot x \\ x \cdot 1 \rightarrow x & x \cdot (y \cdot z) \leftrightarrow y \cdot (x \cdot z) \\ 1 \cdot x \rightarrow x & (x \cdot y) \cdot z \rightarrow x \cdot (y \cdot z) \\ (x^-)^- \rightarrow x & x \cdot (x^- \cdot z) \rightarrow z \\ x \cdot x^- \rightarrow 1 & (y \cdot x)^- \rightarrow x^- \cdot y^- \end{array}$$

Those equations used in both directions have a two-headed arrow. To decide validity of an equation $s = t$, the variables in s and t are replaced by distinct new (Skolem) constants, obtaining a ground instance $s' = t'$ of the equation $s = t$. The lexicographic path ordering is extended to a total ordering that includes any constants appearing in s' or t' . Double-headed rules are then used only in a direction that reduces in this ordering. The equation is valid iff both s' and t' have the same normal form. The reason is that ordered rewriting using the above rules causes ground terms to associate to the left and sorts any ground terms so that small terms occur leftmost and constants and their inverses appear together.

6.8. THEOREM (Bachmair et al. 1989). *Suppose R is a finite convergent system for axioms E and $>$ is a completable simplification ordering for which all rules in R decrease. Any fair completion strategy for $>$ will generate a finite convergent system for E (not necessarily identical to R).*

See also [Devie 1990, Bachmair and Dershowitz 1994].

If R is a reduced convergent system and the strategy performs all compositions and collapses, then, by Theorem 5.21, completion will actually produce R .

By adding the following rule, a system will always be found if one exists for any given reduction ordering, even if the ordering is not completable to a strict monotonic well-ordering of ground terms:

Double: Create one rule to take two steps:

$$\frac{E, R, s \rightarrow t[u], l \rightarrow r}{E, R, s \rightarrow t[u], l \rightarrow r, s\mu \rightarrow t[r]\mu} \quad \mu = \text{mgu}(l, u), \text{ nonvariable } u$$

6.9. THEOREM (Bofill, Godoy, Nieuwenhuis and Rubio 1999). *Suppose R is a canonical system for axioms E and $>$ is a reduction ordering for which $R \subseteq >$. With any fair strategy, using the rules deduce, double, orient, and delete, the set of persisting rules will include R .*

The efficiency of completion depends on the number of critical pairs deduced. Simplification (rewriting to normal form), as employed in the above procedure, is one very effective mechanism for eliminating superfluous equations: if an equation or rule can be rewritten to something previously generated, then it is not needed.

We have not specified how to choose which critical pair to consider at any given time in completion, except that it should be done fairly. Generating them in age-order is one possibility. Another idea is to overlap the two smallest equations that have not been tried together so far. This is good because it tends to produce small rules, and small rules tend to be more useful since they are likely to apply more often. It might also be useful to look for equations with relatively many (linear) variables. Using small equations is fair since there are only finitely many equations of a given size or smaller. Lescanne [1984] suggested running completion for a while, then filtering out the most interesting equations (typically small equations), adding them to the original set R , and repeating the process. “Filter and reuse” tends to focus attention on the more interesting equations, and can produce good results.

In order to perform completion, it is necessary to choose a reduction ordering. Since such orderings abound, this can be difficult. In [Plaisted 1986], a method is proposed to circumvent this problem. There, nondeterministic versions of completion are given; the nondeterminism has to do with how a critical pair $s = t$ is converted into a rule, whether $s \rightarrow t$ or $t \rightarrow s$ is chosen. The idea is to rely on Theorem 4.16 to give a simple criterion for nontermination; for any system R satisfying this criterion, there is no simplification ordering $>$ such that for all rules $l \rightarrow r$ in R , $l > r$. When this happens, some nondeterministic choice has to be redone, and completion is again attempted. This method is guaranteed to be able to generate any system that could be generated using a simplification ordering to orient the critical pairs. Another approach is to test whether a member of a particular class of simplification orderings orients a given set of rules, using algorithms for existential fragments of fully-invariant orderings, mentioned in Section 4.

Various techniques have been used in practice to check for redundancy of equations, so as to limit the critical pairs that are required for completeness of completion. Suppose the proof ordering \gg has the property that a proof decreases

by replacing a subproof with one involving terms that are all smaller vis-à-vis the reduction ordering $>$ supplied to completion. Then a critical pair $s = t$ is redundant if there exists an equational proof $s \leftrightarrow u_1 \leftrightarrow \dots \leftrightarrow u_n \leftrightarrow t$, $n \geq 1$, such that term from which the critical pair was formed is greater than each of the intermediate u_i . For the Group Fragment (5.16), the critical pair $(-0) + y = y$, obtained by rewriting $(-0) + (0 + y)$ with $0 + x \rightarrow x$ and $(-x) + (x + y) \rightarrow y$, is redundant since $(-0) + (0 + y)$ is greater than each of the terms in the alternative proof $(-0) + y \rightarrow 0 + y \rightarrow y$. In particular, a critical pair can be ignored if the variable part of either of the rules involved becomes reducible. Some such redundancy criteria have been suggested in [Winkler and Buchberger 1983, Bachmair and Dershowitz 1988, Kapur, Musser and Narendran 1988, Zhang and Kapur 1990]. These criteria can save some work, but experimental results suggest that most do not improve running times significantly, except in the associative-commutative case of Section 7, where it can make a dramatic difference.

We illustrate another phenomenon with the following single axiom for group theory [Higman and Neumann 1952]:

$$x / (((x/x)/y)/z) / (((x/x)/x)/z) \rightarrow y$$

Here x/y is analogous to xy^{-1} . During completion, the unorientable equation $x/x = y/y$ is obtained, but it is clear that the value of x/x does not depend on x , since x does not appear on the right-hand side. So, rather than using this equation wildly, we can introduce a new constant e and the definitional rule $x/x \rightarrow e$. The same situation occurs when completing the Loop axioms. In general, if an equation $r = s$ is derived, and at least one variable appears on only one side of this equation, then we can add rules $r \rightarrow f(x_1, \dots, x_n)$ and $s \rightarrow f(x_1, \dots, x_n)$ where the x_i are all the variables in common to r and s and f is a new function symbol. Knuth and Bendix [1970] suggested treating unorientable equations by introducing such new operators, but the technique often degenerates by coming up with infinitely many new operators. In some cases when completion does not terminate, patterns in the infinite set of rules generated may be observed and used to generate “meta-rules” that encode these patterns. This may permit construction of a convergent “meta-term” rewriting system; see [Kirchner and Hermann 1990].

In principle, any theory with decidable word problem can be solved by rewriting with an ordered system for some conservative extension of the theory [Dershowitz, Marcus and Tarlecki 1988]. This is not, however, true for ordinary rewriting [Kapur and Narendran 1985b].

Completion can be applied to program generation [Dershowitz and Reddy 1993]. This entails obtaining a program in the form of a set of equations by completing an axiomatic specification.

7. Relativized Rewriting

Many axioms are difficult to handle by rewriting. One example is the commutative axiom $x + y = y + x$ which is nonterminating no matter how it is oriented as a

rewrite rule. We can use ordered rewriting, as in Section 6, and restrict application of the rule to commute only in the direction that decreases the term in some given ordering. For example, we might allow $2 + 1$ to be replaced by $1 + 2$, but not vice-versa. However, if an operator like $+$ is associative and commutative, then there are many equivalent ways to represent terms like $a + b + c + d$, which imposes a burden in storage and time on completion.

In the previous section, we saw (Example 6.7) how to use ordered rewriting to decide validity in free Abelian groups. An alternative approach is to apply commutativity only to enable the application of other rules. For example, we would apply a rule $x \cdot 1 \rightarrow x$ to $a \cdot 1$, as well as to $1 \cdot a$. This “relativized” rewriting permits one to treat such axioms in a special way, without explicitly representing many equivalent forms of a term. The cost is a more complicated rewriting relation, more difficult termination proofs, and a more complicated completion procedure.

The general idea is to let individual terms stand for their E -equivalence classes, for some equational theory E . For example, if E includes associative and commutative (AC) axioms for \star , then the terms $(a \star b) \star a^-$, $a^- \star (a \star b)$, $a \star (b \star a^-)$, etc., will all be rewritable by $x \star x^- \rightarrow 1$. Usually some representation of the whole equivalence class is used; thus it is not necessary to store all the different terms in the class.

To define a rewriting relation on E -equivalence classes, we use the equivalence relation $s =_E t$, defined as $E \models s = t$. If s is a term, let $[s]_E$ be its E -equivalence class, containing all terms E -equivalent to s . The simplest approach would be to say that $[s]_E \rightarrow [t]_E$ if $s \rightarrow t$. Retracting this back to individual terms, we say that $u \rightarrow_{R/E} v$ if there are terms s and t such that $u =_E s$, $s \rightarrow_R t$, and $v =_E t$. This relation R/E is called a *class rewriting system*; however, R/E rewriting is difficult to compute, even when equivalence classes are finite, since it requires searching through all of $[u]_E$ for potential redexes. Note that E -equivalent rules are redundant, in this approach.

A computationally simpler idea is to consider the equivalence classes of instances of left-hand sides and rewrite $u \rightarrow v$ if u has a subterm s such that $s =_E l\sigma$ and $l \rightarrow r \in R$ and $v \equiv u$ with s replaced by $r\sigma$. In this case, we write $u \rightarrow_{E \setminus R} v$ and call the relation $E \setminus R$ the *extended rewrite system* for R modulo E . In the associative-commutative case, this means that $u[s]$ rewrites to $u[r\sigma]$ whenever s is equal under AC to an instance $l\sigma$ of the left-hand side of some rule $l \rightarrow r$ (that is, s and $l\sigma$ may have arguments to AC symbols permuted). Thus, AC-matching must be used to detect applicability of extended AC rewriting rules. (Matching a left-hand side l to a term s is essentially unification of the two terms in the theory of E , with the variables of s regarded as new constants that may not be instantiated.) Consider the systems R/E and $E \setminus R$, where R has $a \star b \rightarrow d$ and E is associativity and commutativity of the operator. Then $(a \star c) \star b \rightarrow_{R/E} c \star d$, since $(a \star c) \star b =_E c \star (a \star b)$. However, it is not true that $(a \star c) \star b \rightarrow_{E \setminus R} c \star d$ since there is no subterm of $(a \star c) \star b$ that is E -equivalent to $a \star b$. On the other hand, $(b \star a) \star c \rightarrow_{E \setminus R} d \star c$, since $b \star a =_E a \star b$.

The extended rewrite relation only requires using the equational theory on the chosen redex s instead of the whole term to “semantically” match s with the left-

hand side of some rule. Such E -matching is often easy enough computationally to make $E \setminus R$ rewriting much more efficient than R/E rewriting, but semantic matching is undecidable in general (Hilbert's Tenth Problem being a special case). Construction of convergent systems using this rewriting relation requires semantic unification algorithms (see [Baader and Snyder 2001], Chapter 8 of this Handbook). This, very successful approach (to the problem that Knuth left open) was initiated by Lankford and Ballantyne [1977] and Peterson and Stickel [1981] and generalized in [Jouannaud and Kirchner 1986, Bachmair and Dershowitz 1989]. It applies to theories with permutative axioms like commutativity, AC, or AC with idempotence and/or identity.

It is impossible for class rewriting to be confluent in the traditional sense, if E equivalence classes are nontrivial, since any term E -equivalent to a normal form will also be a normal form of a term. Instead, to capture the property that class rewriting is confluent when considered as a rewrite relation on equivalence classes, we say that R/E is (*class*) *confluent* if for any term t , if $t \rightarrow_{R/E}^* u$ and $t \rightarrow_{R/E}^* v$ then there are u' and v' such that $u \rightarrow_{R/E}^* u'$, $v \rightarrow_{R/E}^* v'$, and $u' =_E v'$. If R/E is class confluent and terminating then a term may have more than one normal form, but all of them will be E -equivalent. Furthermore, if R/E is class confluent and terminating, then we can reduce any $R = \cup E$ -equivalent terms to E equivalent terms by rewriting. Then an E -equivalence procedure can be used, if there is one.

Though $E \setminus R$ rewriting is much more efficient than R/E rewriting, $\rightarrow_{R/E}$ has better logical properties for deciding $R \cup E$ equivalence. So the theory of relativized rewriting is largely concerned with finding connections between these two rewriting relations.

7.1. DEFINITION (*Church-Rosser Modulo* [Peterson and Stickel 1981]). The rewrite relation $E \setminus R$ is *Church-Rosser modulo E* if any two $R = \cup E$ -equivalent terms can be $E \setminus R$ rewritten to E -equivalent terms.

This is not the same as saying that R/E is Church-Rosser, considered as a rewrite system on E -equivalence classes.

It follows that

7.2. THEOREM (Peterson and Stickel 1981). *Let R be a rewrite system and E an equational system. If all of the following hold:*

1. R is finite,
2. $\rightarrow_{R/E}$ is terminating,
3. R is Church-Rosser modulo E ,
4. E -matching is solvable,

then equivalence in the combined theory of $R = \cup E$ is decidable, by normalizing with $\rightarrow_{E \setminus R}$.

Note that $\rightarrow_{E \setminus R}$ is a subset of $\rightarrow_{R/E}$, so if R/E is terminating, so is $E \setminus R$. But Church-Rosser modulo E is not a local property; as stated, it's not obvious that the property is decidable.

Suppose R is a binary relation, E is an equivalence relation, and T is a binary relation (think of $\rightarrow_{E \setminus R}$) lying between R and R/E , that is, $R \subseteq T \subseteq R/E$. We say that T is *Church-Rosser modulo E* with R if any two $R \cup E$ -equivalent elements can be rewritten by T into E -equivalent elements.

7.3. DEFINITION (*Local Coherence [Jouannaud and Kirchner 1986]*). A binary relation \rightarrow_T is *locally coherent* with a binary relation R modulo a binary relation E if $\leftarrow_T \circ \leftrightarrow_E$ is contained in $\rightarrow_T^* \circ \leftrightarrow_E^* \circ \leftarrow_T^*$.

The idea of coherence is that there should be some similarity in the way different elements of an E -equivalence class rewrite.

7.4. LEMMA (*Coherence [Jouannaud and Kirchner 1986]*). Let \rightarrow_T be a relation such that $\rightarrow_R \subseteq \rightarrow_T \subseteq \rightarrow_{R/E}$. Suppose $\rightarrow_{R/E}$ is terminating. Then \rightarrow_T is Church-Rosser modulo E with R iff \rightarrow_T is locally coherent modulo E with both \rightarrow_R and \leftrightarrow_E .

This lemma reduces the desired Church-Rosser property to local properties that can be tested, analogous to the Diamond Lemma (5.13). It forms the basis of methods for completing R and E to obtain systems that are Church-Rosser modulo E .

Another approach is to add rules to R to obtain a logically equivalent system S/E ; that is, $R \cup E$ and $S \cup E$ have the same logical consequences (that is, are equivalent), but $E \setminus S$ rewriting is the same as R/E rewriting. Therefore we can use the computationally simpler $E \setminus S$ rewriting to decide the equality theory of R/E . This is done for associative-commutative operators by Peterson and Stickel [1981]. In this case, confluence can be decided by methods simpler than those above. Consider the special case of rewriting relative to the associative and commutative axioms $E = \{f(x, y) = f(y, x), f(f(x, y), z) = f(x, f(y, z))\}$ for a function symbol f . In this case, one can *flatten* the term structure so that $E \setminus R$ rewriting can be used rather than R/E rewriting, that is, a term $f(s_1, f(s_2, \dots, f(s_{n-1}, s_n) \dots))$, where none of the s_i have f as outermost function symbol, is represented as $f(s_1, s_2, \dots, s_n)$, where f is now a varyadic symbol, taking a variable number of arguments. For such flattened terms, all permutations of arguments of f are identified. This means that each AC -equivalence class is represented by one flat term. Since now all members of a given AC -equivalence class have the same term structure, $AC \setminus R$ rewriting is easier. However, the subterm structure has been changed; $f(s_1, s_2)$ is a subterm of $f(f(s_1, s_2), s_3)$ but there is no corresponding subterm of $f(s_1, s_2, s_3)$. Thus, $AC \setminus R$ rewriting does not simulate R/AC rewriting on the original system. For example, consider $R = \{a \star b \rightarrow d\}$ with AC axioms for \star . Suppose $s \equiv (a \star b) \star c$ and t is $d \star c$. Then $s \rightarrow_{R/AC} t$; in fact, $s \rightarrow_{AC \setminus R} t$. However, if we flatten the terms, then s becomes $\star(a, b, c)$ and s no longer rewrites to t since the subterm $\star(a, b)$ has disappeared. To overcome this, we must add *extensions* to rewrite rules to simulate their effect on flattened terms. The extension of the rule $a \star b \rightarrow d$ is $x \star a \star b \rightarrow x \star d$, where x is a new variable. With this extended rule, we do have that $a \star b \star c \rightarrow_{AC \setminus R} d \star c$.

The general idea, then, is to flatten terms, and extend R by adding extensions of rewrite rules to it. Then, extended rewriting on flattened terms using the extended R is equivalent to class rewriting on the original R . Formally, suppose s and t are terms and s' and t' are their flattened forms. Suppose R is a rewrite system and S is R with the extensions added. Suppose E is associativity and commutativity. Then $s \rightarrow_{R/E} t$ iff $s' \rightarrow_{S \setminus E} t'$. The extended R is obtained by adding, for each rule of the form $f(r_1, \dots, r_n) \rightarrow s$ where f is AC, an extended rule of the form $f(x, r_1, \dots, r_n) \rightarrow f(x, s)$, where x is a new variable. The original rule is also retained. This idea does not always work on other equational theories, however. Note that some kind of AC matching is needed for extended rewriting. This can be fairly expensive, since there are so many permutations to consider, but it is fairly straightforward to implement. Completion relative to AC can be done with the flattened representation [Peterson and Stickel 1981], using AC matching and unification. Unification is needed to form critical pairs $\mathbf{cp}_{AC}(E)$ modulo AC. This means that we look at the finite set of most general substitutions that allow one side of an equation to be AC-equivalent to a nonvariable subterm of one side of another equation, and get a critical pair for each such ambiguously rewritable term.

Specifically, the completion procedure of Section 6 may be modified in the following ways to handle AC symbols:

1. AC-unification is used to generate critical pairs instead of ordinary unification:

$$\frac{E, R}{E, R, s = t} \quad \text{if } s = t \in \mathbf{cp}_{AC}(E \cup R)$$

2. An additional expansion operation is needed whenever a new equation $f(s, t) = r$ is formed, where f is an AC symbol:

$$\frac{E, R, f(s, t) = r}{E, R, f(s, t) = r, f(s, f(t, z)) = f(r, z)} \quad \text{if } f \text{ is AC, } f(s, t) \not\prec r$$

for some new variable z . This **extension** rule ensures that $f(s, t) = r$ can be used even when rearrangement is needed to get an instance of its left-hand side.

3. The ordering must be such that $s > t$ only if $s' > t'$ for all AC variants s' of s and t' of t . This ensures that each new AC-rule reduces the term it is applied to:

$$\frac{E, R, l = r}{E, R, l \rightarrow r} \quad \text{if } l > r$$

4. Any equation between AC-variants is deleted:

$$\frac{E, R, s = t}{E, R} \quad \text{if } s =_{AC} t$$

5. AC-rewriting is used for simplification:

$$\frac{E, R, l = r}{E, R, l = r'} \quad \text{if } r \rightarrow_{R/AC} r' \qquad \frac{E, R, l \rightarrow r}{E, R, l \rightarrow r'} \quad \text{if } r \rightarrow_{E \cup R/AC} r'$$

$$\frac{E, R, s[l\sigma] = t}{E, R, s[r\sigma] = t} \quad \text{if } l \rightarrow_{AC \setminus E} r, l \text{ not a variant of } s$$

$$\frac{E, R, s[l\sigma] \rightarrow t}{E, R, s[r\sigma] \rightarrow t} \quad l \rightarrow_{AC \setminus E \cup R} r, l \text{ not a variant of } s$$

Recall that the Chameleons (Example 1.3) do not terminate. But if we turn any one of the rules around, they do! Though the result is not confluent, AC completion can be used to generate a confluent system of rules and equations: Start off with the rules as unoriented equations, and use an ordering with $R > Y > G$. The oriented equations are:

$$\begin{aligned} R Y &\rightarrow G G \\ R R &\rightarrow G Y \\ R G &\rightarrow Y Y \end{aligned}$$

Notice how the second rule acts contrary to the “real” chameleons. The flattened, extended rules are:

$$\begin{aligned} Y R z &\rightarrow G G z \\ R R z &\rightarrow G Y z \\ G R z &\rightarrow Y Y z \end{aligned}$$

and their commutative variants. The GR and extended YR rules produce a critical pair $YYY = GGG$, which gets oriented from left to right: $YYY \rightarrow GGG$. The complete system reduces the initial state and the monochrome states to distinct normal forms. (Which?) Since the system is Church-Rosser, there is no way to get from the initial arrangement of chameleons to one in which they are of uniform color, no matter which way any of the rules are used.⁷

7.5. EXAMPLE (*Abelian Groups II*). The AC-completion procedure, given

$$x \cdot 1 = x \quad x \cdot x^- = 1$$

where \cdot is AC, and a polynomial ordering in which $\tau(x \cdot y) = \tau(x) + \tau(y) + 1$, $\tau(x^-) = 2\tau(x)$, and $\tau(1) = 1$, generates the following decision procedure for Abelian groups:

$$\begin{aligned} 1^- &\rightarrow 1 & x \cdot 1 &\rightarrow x \\ (x^-)^- &\rightarrow x & (y \cdot x)^- &\rightarrow x^- \cdot y^- \\ x \cdot x^- &\rightarrow 1 & x \cdot (x^- \cdot z) &\rightarrow z \end{aligned}$$

The last rule is a composed version of the extension $x \cdot (x^- \cdot z) \rightarrow 1 \cdot z$ of $x \cdot x^- \rightarrow 1$. This extended rule, together with $(y_0 \cdot x_0)^- \rightarrow x_0^- \cdot y_0^-$, forms a critical pair

$$(x_2 \cdot z_1)^- (x_2 \cdot (x_1 \cdot x_2)^- \cdot z_2)^- = (z_1 \cdot z_2)^-$$

⁷The rewriting solution to the Chameleon puzzle is due to Claude Marché.

via AC-unifier $\{x \mapsto x_1 \cdot x_2, z \mapsto z_1 \cdot z_2, y_0 \mapsto x_1 \cdot z_1, x_0 \mapsto x_2 \cdot x^- \cdot z_2\}$. Both sides reduce to $z_2^- \cdot z_1^-$. With this system, both sides of any identity reduce by AC-rewriting to AC-equivalent terms.

The Grecian Urn (1.2) may also be viewed as an AC-rewriting system. Beans are rearranged until the pair of beans to be removed are adjacent. With extended rules this system is confluent and terminating. For instance, $BW \rightarrow W$ and the extended rule $WWz \rightarrow Bz$, rewrite a BWW bean arrangement to WW and BB , respectively, both of which rewrite to B . This means that the normal form is independent of the order in which rules are applied. If there are an even number of white beans, they can be paired and reduced to black, and then all the black beans reduce to one; if there are an odd number, the leftover white bean swallows all remaining blacks. Thus rewriting solves the problem since the color of the last bean is determined by the normal form of the rewriting relation, which does not depend on which rewrite rules are chosen.

7.6. EXAMPLE (*Ring Idempotents*). The standard ring axioms plus

$$\begin{aligned} aa &= a & bb &= b & cc &= c \\ (a+b+c)(a+b+c) &= a+b+c \end{aligned}$$

can be completed, with an appropriate ordering, to a convergent AC system that includes the equations

$$\begin{aligned} ba &= -ab - bc - cb - ac - ca \\ bca &= abc + acb + cab + cac + cbc + ab + ab + ac + ac + cb + cb + bc + ca \end{aligned}$$

The normal forms of this system include (besides inverses and sums) all monomials not containing aa , bb , cc , ba , or bca (the order of factors matters since multiplication is not commutative).

Lankford and Ballantyne [1977] extended completion to handle the most important nonterminating axioms and found the above associative-commutative systems for free Abelian groups; Peterson and Stickel [1981] used a similar process to derive convergent systems for free commutative rings with unit and distributive lattices (and tried Boolean algebra without success). Hullot [1980] used these procedures to derive systems for various quasigroups, associative and non-associative rings (but could not handle anticommutative or Lie rings), left and right A -modules, A -bimodules, A -rings, and A -algebras.

An alternative approach in which E is only used to check “semantic equality” of subterms matching different occurrences of the same non-left-linear rule variable was proposed by Pedersen [1985a]. In general, (infinitely) many more rules may be needed to complete the system. The more general issue of deduction modulo a non-equational theory E is raised in [Dowes, Hardin and Kirchner 1994].

Termination for relativized rewrite systems is tricky to test for; one needs special termination orderings. The problem is that E -equivalent terms are identified

when doing relativized rewriting, so that it pays to make all E -equivalent terms equivalent in the quasi-ordering. This is problematic for the recursive path ordering and similar orderings. For example, we can represent associative-commutative equivalence classes by flattened terms, as mentioned above. However, applying the multiset path ordering to such terms violates monotonicity. For example, suppose $\times > +$ and \times is associative-commutative. Then $x \times (y + z) > x \times y + x \times z$. By monotonicity, we should have $u \times x \times (y + z) > u \times (x \times y + x \times z)$. But, in fact, the term on the right is larger in the multiset path ordering. A number of attempts have been made to overcome this, starting with the “associative path ordering” of Dershowitz, Hsiang, Josephson and Plaisted [1983]. This ordering applied to transformed terms, in which big operators like \times were pushed below small operators like $+$. More recently, better orderings have been devised [Bachmair and Plaisted 1985, Rubio and Nieuwenhuis 1993, Kapur and Sivakumar 1997, Rubio 1999].

8. Equational Theorem Proving

Huet [1981] showed how the completion procedure serves as a semidecision procedure for validity in equational theories, as long as no unorientable simplified critical pair is generated. Lankford [1975] proposed that completion-like methods be incorporated in resolution-based theorem provers for the first-order predicate calculus, with paramodulation used for unorientable equations.

Indeed, it follows from the completeness of the completion process (Theorem 6.6) that a rewrite proof between two ground terms will eventually be generated by ordered completion from a theory iff the terms are equal in the theory. Ordered completion uses paramodulation to avoid the possibility of failure inherent in the original [Knuth and Bendix 1970, Huet 1981] procedures. Thus, the uniform word problem in arbitrary equational theories can always be semidecided by running ordered completion using a completable simplification ordering:

8.1. THEOREM (Hsiang and Rusinowitch 1991, Bachmair and Dershowitz 1994). *Suppose $s = t$ is a theorem of E . For any fair strategy using the completion rules, starting with E , and a completable simplification ordering, eventually s and t will rewrite to the identical term using the generated rules and equations.*

With an empty ordering, completion amounts to ordinary paramodulation of unit equations; with more of an ordering, completion can be more effective—by reducing the number of allowed inferences and increasing the amount of simplification that may be performed without loss of completeness. The Gröbner basis approach to properties of polynomial ideals and solving word problems [Becker and Weispfenning 1993] is similar to completion.

8.2. EXAMPLE (*Distributive Lattices*). With axioms:

$$\begin{array}{ll}
x \sqcap x & = x & x \sqcup x & = x \\
x \sqcap y & = y \sqcap x & x \sqcup y & = y \sqcup x \\
(x \sqcap y) \sqcap z & = x \sqcap (y \sqcap z) & (x \sqcup y) \sqcup z & = x \sqcup (y \sqcup z) \\
(x \sqcup y) \sqcap x & = x & (x \sqcap y) \sqcup x & = x \\
x \sqcup (y \sqcap z) & = (x \sqcap y) \sqcup (x \sqcap z)
\end{array}$$

and a lexicographic path ordering that makes meet bigger than join, completion eventually generates:

$$x \sqcap (y \sqcup z) \rightarrow (x \sqcup y) \sqcap (x \sqcup z)$$

Simplifying by ordered rewriting (*ordered simplification*) has been implemented (e.g. [McCune 1989]). Its advantage is that it is not necessary to use a special unification algorithm for associative and commutative functions. A disadvantage is that it is often necessary to keep more terms than if an associative-commutative unification algorithm were used. The idea of ordered simplification was taken further, to “constrained completion”, in [Kirchner, Kirchner and Rusinowitch 1990, Martin and Nipkow 1990, Peterson 1990]. The idea is to constrain rewrite rules by inequalities between (substitution instances of) variables. Thus we can have a rule $x \cdot y \rightarrow y \cdot x$ with the constraint that $x > y$ in the given ordering. They present methods for showing that such constrained rewriting systems are ground confluent.

8.3. EXAMPLE (*Bags*). The following system normalizes any ground term over the equational theory containing the axioms of associativity and commutativity by performing ordered simplification with the lexicographic path ordering:

$$\begin{array}{ll}
(xy)z & \rightarrow x(yz) \\
xy & \leftrightarrow yx \\
x(yz) & \leftrightarrow y(xz)
\end{array}$$

Using this system, the term $(bc)a$ will rewrite to $b(ca)$ using $(xy)z \rightarrow x(yz)$, then to $b(ac)$ using ordered simplification and the equation $xy = yx$, then to $a(bc)$ using the equation $x(yz) = y(xz)$ and ordered simplification.

Other systems like this that are convergent with respect to ordered simplification are given in [Martin and Nipkow 1990].

Lescanne [1984] experimented with various presentations of groups: applying Knuth’s idea of handling unorientable equations by introducing new operators to Higman and Neumann’s one-equation presentation, completion came up with definitions of identity and inverse. Pedersen [1984a, 1985a] used this technique, plus a special way of handling permutative axioms, to generate systems for some entropic groupoids, and also solved some one relation monoid word problems [Pedersen 1989]. Christian [1989] was partially successful in his work on Burnside groups and Grau’s

ternary Boolean algebra. Foret [1988] found rewrite-based decision procedures for several systems (K, Q, T, S5) of propositional modal logic. Edelson [1990] was able to rediscover syntactic proofs for properties of regular rings.

8.4. EXAMPLE (*Groupoid II*). The entropic groupoid [Pedersen 1984b]

$$(xy)(zw) = (xz)(yw) \quad (xy)x = x$$

has the following decision procedure for ordered simplification, where \star is defined by the equation $x \star z = (xy)z$, cancelling y (on the basis of the inferred equation $(xy)z = (xy')z$):

$$\begin{array}{ll} x \star x \rightarrow x & x(y \star z) \rightarrow xz \\ (xy)z \rightarrow x \star z & x \star (yz) \rightarrow x \star z \\ x(yz) \rightarrow xz & (x \star y) \star z \rightarrow x \star z \\ (x \star y)z \rightarrow xz & x \star (y \star z) \rightarrow x \star z \\ (xy) \star z \rightarrow xz & (xy)(zw) \leftrightarrow (xz)(yw) \end{array}$$

An *absorbing* rule is one in which every nonvariable subterm on the left contains all the variables [Pedersen 1985b], ensuring that its critical pairs with ground equations are still ground. With associative-commutative systems this is essentially never the case, but sometimes the nonground pairs are certain to reduce to ground ones. Ballantyne and Lankford [1981] accordingly gave rewriting-based procedures for the uniform word problem in finitely-generated commutative semigroups; with Butler [1984], they did the same for finitely-generated Abelian groups; and Lankford [1980] used this method to show decidability of the uniform word problem in finitely-presented J -algebras. Pedersen [1985b] gave sufficient syntactic conditions for decidability of the uniform word problem for some absorbing systems, including finitely-presented loops, using a completion procedure that adds new symbols as needed. Even when the uniform word problem is undecidable, as for non-Abelian groups, this method frequently finds decision procedures for specific word problems. In this vein, Le Chenadec [1985] did substantial work on finitely-presented groups from topology and geometry, including the Coxeter groups (showing termination was sometimes difficult).

Burris and Lawrence [1991] presented systems for finite fields, rings with $x^n = x$, and such rings with n prime and $nx = 0$ (studied also by Nipkow [1990]). Kapur and Zhang [1989] used an enhanced associative-commutative completion procedure (which avoids many redundant critical pairs) to prove commutativity for rings with $x^n = x$, for many specific n . Anatharaman and Hsiang [1990] used a combination of ordered and associative-commutative completion to derive purely syntactic proofs of the Moufang identities for alternative rings. Pedersen [1985a] used a variant of associative-commutative completion to generate an infinite system that “computes” Whitman normal form for lattices. Finally, the Robbins algebra conjecture has recently been proved using rewriting techniques [McCune 1997]. The 12-step equational proof followed after 50,000 equations were generated and 6,000

simplification steps were performed; almost 90% of the computer's effort went into simplification. Simplification was critical for the success of the automated proof.

As first suggested in [Hsiang and Dershowitz 1983], one can apply completion to full first-order theorem proving, by using Boolean rings [Zhegalkin 1927, Stone 1936] to represent formulæ:

$$\begin{array}{ll}
 x \cdot T & \rightarrow x & x \oplus F & \rightarrow x \\
 x \cdot F & \rightarrow F & (x \oplus y) \cdot z & \rightarrow (x \cdot z) \oplus (y \cdot z) \\
 x \cdot x & \rightarrow x & x \cdot x \cdot y & \rightarrow x \cdot y \\
 x \oplus x & \rightarrow F & x \oplus x \oplus y & \rightarrow y
 \end{array}$$

where \cdot is "conjunction", \oplus is "exclusive or", and both are associative and commutative. With this system, all propositional tautologies reduce to T and contradictions to F . To prove validity of a formula, one Skolemizes its negation (and renames variables) to obtain a universally quantified formula, and expresses it using the above connectives. Then, were the original formula true, there would be an equational proof of the contradiction $T = F$. Completion can be used to discover such a proof.

As a very simple example, consider the theorem

$$[\exists x p(x) \wedge \forall x (p(x) \Rightarrow p(f(x)))] \Rightarrow \exists x p(f(f(x)))$$

Its Skolemized negation is equivalent to the following equations:

$$\begin{array}{ll}
 p(f(f(x))) & = F \\
 p(a) & = T \\
 p(x) \vee p(f(x)) & = p(f(x))
 \end{array}$$

where a is a Skolem constant. These equations entail the contradiction:

$$\begin{aligned}
 T &= T \vee [p(f(a)) \vee p(f(f(a)))] \\
 &= p(a) \vee [p(f(a)) \vee p(f(f(a)))] \\
 &= [p(a) \vee p(f(a))] \vee p(f(f(a))) \\
 &= p(f(a)) \vee p(f(f(a))) \\
 &= p(f(f(a))) = F
 \end{aligned}$$

This method can be refined [Hsiang 1985] to ignore many critical pairs, but one must take care to ensure completeness. Further work along these lines includes [Bachmair and Dershowitz 1987, Zhang 1994, Kapur and Narendran 1985a], see also [Bachmair and Ganzinger 2001] (Chapter 2 of this Handbook). For the use of Boolean rings in the propositional case, see [Hsiang and Huang 1996, Dershowitz, Hsiang and Shieng 2000].

There is a subtlety in the notion of logical consequence which should be mentioned here. A first-order structure M may include elements that are inaccessible, and cannot be represented by any ground term. For example, if E includes the constant 0 and unary function symbol s , then the ground terms are

$\{0, s(0), s(s(0)), \dots\}$. However, the domain of the structure M may include elements d that cannot be represented by any of these elements; formally, it may be that $d \neq 0^M, d \neq s(0)^M, d \neq s(s(0))^M, \dots$. This means that the notion of logical consequence may not agree with naïve intuition, where we wish to infer those equations $s = t$ such that whenever the variables of s and t are replaced by ground terms in a systematic way, the result is a consequence of E . We are often interested in those equations $s = t$ such that all ground instances are logical consequences of E . For example, suppose E contains two axioms: $x + 0 = x$, and $x + s(y) = s(x + y)$. These equations completely define the usual addition operation on nonnegative integers, regarding s as the successor function. We expect addition to be commutative, which means that we expect to have $E \models u + v = v + u$. Indeed, any ground instance of this equation containing only the function symbols s and 0 is a logical consequence of E . However, the equation $u + v = v + u$ itself is *not* a logical consequence of E , since it is possible to construct a structure M , containing inaccessible elements, in which E is valid but the equation $u + v = v + u$ is not. The *inductive theory* of a set E of equations is the set of equations $s = t$ such that all ground instances $s\sigma = t\sigma$ are logical consequences of E . These two concepts are related as follows: Consider the term algebra $\mathcal{T}(\mathcal{X})$ over a (countable) set \mathcal{X} of variables, for some vocabulary. Then $\mathcal{T}(\mathcal{X})/\equiv_E \models s = t$ iff $s = t$ is a logical consequence of E . And $\mathcal{T}(\emptyset)/\equiv_E \models s = t$ iff $s = t$ is in the inductive theory of E . The quotient set $\mathcal{T}(\mathcal{X})/\equiv_E$ is a *free algebra* in the class of models of E and $\mathcal{T}(\emptyset)/\equiv_E$ is the *initial algebra* in this class.

In fact, by Gödel's First Incompleteness Theorem, there can be no method of deriving all the quantified formulæ of arithmetic (addition, multiplication, and inequality) that are true of the nonnegative integers. This means that every set of axioms for the integers has a nonstandard model, containing inaccessible elements, in which some inductive theorem is false in the model. Another way of looking at this is that it is impossible in a finite axiom system to fully characterize the integers, or the set of finite terms. Nevertheless, there are incomplete methods for deriving and verifying equations in an inductive theory, which involve some form of mathematical induction. A modified completion procedure is an integral part of many of the inductionless induction theorem-proving methods [Comon 2001]; see Chapter 14 of this Handbook.

Oftentimes one is interested in the satisfiability of equations. We may want to know if $E \models (\exists x)s = t$, that is, from E does it follow that there is an x such that $s = t$? This topic is covered in [Baader and Snyder 2001] (Chapter 8 of this Handbook), including the rewriting-based approach called "narrowing".

9. Conditional Rewriting

Sometimes equational systems are not expressive enough. For example, we may want to state that $s = t$ if some condition C is true, as in

$$\frac{x}{y} = \frac{x - y}{y} + 1 \quad \text{only if } y \neq 0$$

In the many cases where specifications are naturally conditional, we cannot use conventional term-rewriting systems. Instead, we use *conditional rewrite systems*, in which the rewrite rules have *conditions* attached, which must hold true for the rewrite to transpire. A rule $l \rightarrow r$ with a condition C is written $C \mid l \rightarrow r$, as in

$$y \neq 0 \mid \frac{x}{y} \rightarrow \frac{x-y}{y} + 1$$

where C is viewed as a “guard” for the application of the rewrite. Conditions C may take several different forms; for example, they may be logical formulæ, or equations, or inequations. If C is a logical formula, the meaning is that a term encompassing l rewrites to the corresponding term with the appropriate instance of r in place of l only if C is true. That leaves the question of how to determine whether C is true.

In this chapter, we will assume that C is a conjunction of equations, omitting a condition when it is the empty conjunction. For example,

$$\begin{array}{ll} (x > y) = T & \mid \quad \max(x, y) \rightarrow x \\ (x > y) = F & \mid \quad \max(x, y) \rightarrow y \\ & \quad s(x) > s(y) \rightarrow x > y \\ & \quad s(x) > 0 \rightarrow T \\ & \quad 0 > x \rightarrow F \end{array}$$

define maximum and greater-than for tally numbers $s^i(0)$. The equations in the conditions can be evaluated recursively by conditional rewriting, in a manner similar to the goal-subgoal structure of Prolog, making conditional rewrite systems an attractive combination of logic and functional programming paradigms.

A *conditional equation* (or *equational Horn clause*) is an implication of the form

$$u_1 = v_1 \wedge \cdots \wedge u_n = v_n \Rightarrow l = r$$

In a rule of form $C \mid l \rightarrow r$, we call C the *premiss* and $l \rightarrow r$ the *conclusion*. The semantics is that $C \Rightarrow l = r$. In general, C may be any formula, and may contain the equality predicate, or the rewrite relation \rightarrow , or its variants. If C is a conjunction of equations, we call it a *semi-equational* rule; if C is a conjunction of statements of the form $s_i \downarrow t_i$, we call it a *join rule*.

A (*standard* or *join*) *conditional rewrite system* is a collection of rules of the form

$$u_1 \downarrow v_1 \wedge \cdots \wedge u_n \downarrow v_n \mid l \rightarrow r$$

intending that terms $u[l\sigma]$, containing an instance of left-hand side l , rewrite to $u[r\sigma]$ only when all the conditions $u_i\sigma \downarrow v_i\sigma$ hold for that substitution σ . The most popular operational semantics for such a system require both sides of each condition to rewrite to the same normal form before an instance of l may be rewritten.

9.1. EXAMPLE (*Conditional Append*). To feel the flavor of this version of rewriting, consider the system

$$\begin{array}{llll} \text{null}(\epsilon) & \rightarrow & T & \text{head}(x : y) \rightarrow x \\ \text{null}(x : y) & \rightarrow & F & \text{tail}(x : y) \rightarrow y \\ \text{append}(\epsilon, y) & \rightarrow & y & \\ \text{null}(x) \downarrow F \mid \text{append}(x, y) & \rightarrow & \text{head}(x) : \text{append}(\text{tail}(x), y) & \end{array}$$

and its derivation

$$\text{append}(a : (b : \epsilon), c : \epsilon) \rightarrow^* a : (b : (c : \epsilon))$$

Confluence of conditional systems in general is undecidable [Brand, Darringer and Joyner 1979, Kaplan 1987], unlike the unconditional case. Other approaches to conditional confluence include using oriented conditions, an analogue to orthogonality (Theorem 10.10 below), or restricted-depth proofs of conditions [Dershowitz, Okada and Sivakumar 1987, Giovannetti and Moiso 1987].

We say a conditional system R is *Noetherian* if there are no infinite sequences t_1, t_2, t_3, \dots of terms such that $t_1 \rightarrow_R t_2 \rightarrow_R t_3 \dots$. This corresponds to the concept of termination for unconditional systems, but each rewrite step involves the recursive evaluation of conditions. That is, to determine whether $s \rightarrow_R t$, it may be necessary to determine whether $u \rightarrow_R v$ for u and v obtained from the condition of some rule of R . It is possible for a system to be Noetherian even when such recursive evaluation of conditions does not terminate; in such a case, the computation might not be terminating. In fact, it may happen that the rewrite relation $s \rightarrow_R t$ is undecidable.

Semi-equational systems are confluent if they are Noetherian and all critical pairs are joinable [Dershowitz and Plaisted 1988]. In contradistinction with the unconditional case, only under certain circumstances are Noetherian join systems confluent—even if all their critical pairs are joinable [Dershowitz et al. 1987]. But we first need a suitable notion of critical pair:

9.2. DEFINITION (*Conditional Critical Pair*). Let $>$ be a reduction ordering. The conditional equation $c\mu \wedge p\mu \Rightarrow s\mu[r\mu] = t\mu$ is a *conditional critical pair* of the conditional equations $c \Rightarrow l = r$ and $p \Rightarrow s = t$, if l unifies via most general unifier μ with a nonvariable subterm of s , the ordering is such that $l\mu \not\prec c\mu$, $s\mu \not\prec p\mu$, $l\mu \not\prec r\mu$, $s\mu \not\prec t\mu$, and $c\mu \wedge p\mu$ is satisfiable in E . A conditional critical pair $p \Rightarrow s = t$ is *joinable* if $s\sigma$ and $t\sigma$ are joinable for all σ satisfying p .

By $l\mu \not\prec c\mu$, we mean that no term in $c\mu$ is always greater than $l\mu$. We use $\text{cp}(E)$ to denote the set of conditional critical pairs so defined.

If $c|l \rightarrow r$ and $p|s \rightarrow t$ are two different rules and l and s unify with most general unifier σ , then the critical pair $p\sigma \wedge c\sigma \Rightarrow r\sigma = t\sigma$ is an *overlay* for those rules.

9.3. THEOREM (Dershowitz, Okada and Sivakumar 1988). A *standard Noetherian conditional rewrite system* is confluent if no left-hand side unifies with a nonvariable proper subterm of a left-hand side and every overlay critical pair is joinable.

Such systems with only top-level critical pairs are called *overlaying*. See also [Gramlich and Wirth 1996].

9.4. DEFINITION (*Decreasing System*). A conditional rewrite system R is *decreasing* if there exists a well-founded ordering $>$ containing the rewrite relation \rightarrow_R and which satisfies two additional requirements:

1. $>$ has the subterm property $f(\dots, s, \dots) > s$, and
2. $l\sigma > c\sigma$ for each rule $c \mid l \rightarrow r$ in R and substitution σ .

Decreasing systems exactly capture the finiteness of recursive evaluation of terms; they refine the suggestion in [Kaplan 1987]. If R is decreasing, then the rewriting relation is decidable, since evaluating the conditions involves terms smaller than the redex. It's possible for a conditional rewrite relation to be Noetherian without being decreasing. In practice, simplification orderings $>$ may be used to show the decreasing property. This definition implies that all variables in the premiss appear also in the left-hand side. The notion needs to be extended, therefore, to cover systems (important in logic programming) with variables in conditions that do not also appear in the left-hand side. See [Bertling and Ganzinger 1989, Hanus 1995, Marchiori 1996, Ohlebusch 1999].

Standard join systems are locally confluent if they are decreasing and all critical pairs are joinable.

9.5. THEOREM (Dershowitz, Okada and Sivakumar 1988). *A decreasing system is confluent (hence, convergent) iff there is a rewrite proof of $s\sigma = t\sigma$ for each critical pair $c \Rightarrow s = t$ and substitution σ such that $c\sigma$ holds.*

Conditional Append (9.1) and the Stack system (1.7) are decreasing. Both have one trivially joinable critical pair for which there is no satisfying substitution.

Unlike the situation for completion of ordinary rewrite systems, there is no general-purpose mechanism for obtaining confluent conditional systems. There are some completion-like procedures for conditional equations [Kaplan 1987, Ganzinger 1991, Kounalis and Rusinowitch 1988], which could in some instances provide decision procedures for conditional equational theories (*quasi-varieties*), but, in practice, they do not work well and further research is required.

The free (initial) algebra for n -generators can be computed by completing the axioms and looking at the normal forms of bigger and bigger words until they stabilize. Pedersen [1988] did this for bands with up to three generators. Conditional rewriting, using conditional equations, can sometimes be used to capture an infinite number of unconditional rules in one conditional one; this approach was taken by Siekmann and Szabo [1982] for bands. For a conditional-rewriting approach to associative-commutative-idempotent systems, see [Baird, Peterson and Wilkerson 1989].

In order to obtain decision procedures employing conditional rewriting, we need to consider logical strengths of systems. The reason for this is as follows: Suppose we are given an equational system E (with equational conditions). We want to obtain an equivalent convergent system which can be used to decide the equality theory of E . One way to do this is to convert E into a join system R , and then complete

R to obtain R' which is decreasing, Noetherian, and confluent. Then we know that $R \models s = t$ iff $R' \models s = t$ by the way completion is done, and $R' \models s = t$ iff s and t have the same R' -normal form, since R' is Noetherian and confluent. However, we do not know that $R' \models s = t$ iff $E \models s = t$, which is what we really want. The reason is that we changed the conditions in E to join assertions to obtain R . Therefore, the following theorem is useful:

9.6. THEOREM (Dershowitz, Okada and Sivakumar 1988). *Let R be a confluent and Noetherian standard conditional system, R' , the corresponding semi-equational systems (with conditions changed to equations), and E the corresponding equational system (with right-hand sides changed to equations). Then the following are equivalent:*

1. $E \models u = v$
2. u and v have the same R -normal form
3. u and v have the same R' -normal form.

If the conditions of the theorem hold, then reduction via R can be used to decide the theory of E . Note that typically E , when converted to R , will not be confluent and Noetherian. Therefore, we have to complete R in a way that does not change the corresponding equational theory E .

Completion for conditional systems is trickier than for unconditional systems. The inference rules we present may be classified into three expansion rules and four contraction rules. Contraction rules significantly reduce space requirements, but make proofs of completeness much more subtle. As with ordinary completion, a reduction ordering \succ is used to control simplification (demodulation).

Superposition (that is, oriented paramodulation of positive equational literals) is performed only at nonvariable positions:

$$\text{Superpose: } \frac{E}{E, e} \quad \text{if } e \in \text{cp}(E)$$

Only positive equations are used in this rule, and only in a decreasing direction. Either side of an equation may be used for superposition, but only if it is potentially the largest term involved. Note that the two conditional equations may actually be renamed variants of one and the same equation.

We need, additionally, a rule that paramodulates into maximal negative literals:

$$\text{Narrow: } \frac{E, q \wedge s[l'] = t \Rightarrow u = v}{E, q \wedge s[l] = t \Rightarrow u = v, \quad p\mu \wedge q\mu \wedge s[r]\mu = t\mu \Rightarrow u\mu = v\mu} \quad \text{if } \left\{ \begin{array}{l} p \Rightarrow l' = r \in E, \\ l \text{ is not a variable} \\ \mu = \text{mgu}(l, l') \\ s[l]\mu \not\leq p\mu, q\mu, t\mu, s[r]\mu \end{array} \right.$$

The condition $s\mu \not\leq p\mu$ means that $s\mu$ is not smaller than any side of any equation in $p\mu$. Whenever this or subsequent rules refer to a conditional equation like $q \wedge s = t \Rightarrow u = v$, the intent is that $s = t$ is any one of the conditions and u is either side of the implied equation.

The last expansion rule in effect resolves a maximal negative literal with reflexivity of equals ($x = x$):

$$\text{Reflect:} \quad \frac{E, q \wedge s = t \Rightarrow u = v}{E, q \wedge s = t \Rightarrow u = v, \quad q\mu \Rightarrow u\mu = v\mu} \quad \text{if} \quad \begin{cases} \mu = \text{mgu}(s, t) \\ s\mu \not\prec q\mu \end{cases}$$

The contraction rules all simplify the set of conditional equations:

$$\frac{E, q \Rightarrow u = u}{E} \quad \frac{E, q \wedge s = s \Rightarrow u = v}{E, q \Rightarrow u = v}$$

$$\frac{E, q[l\sigma] \Rightarrow u = v}{E, q[r\sigma] \Rightarrow u = v} \quad \text{if} \quad p \Rightarrow l = r \in E, l\sigma \succ r\sigma, p\sigma, E \models p\sigma$$

$$\frac{E, q \Rightarrow u[l\sigma] = v}{E, q \Rightarrow u[r\sigma] = v} \quad \text{if} \quad \begin{cases} p \Rightarrow l = r \in E, l\sigma \succ r\sigma, p\sigma, E \models p\sigma \\ v \succ u[l\sigma] \vee (v \succ r \wedge l \not\prec u[l\sigma]) \end{cases}$$

The first contraction rule deletes trivial conditional equations. The second allows for deletion of conditions that are trivially true. The last two use decreasing instances to simplify other clauses. One rule simplifies conditions; the other (**compose**) applies to the equation part. In both, the original clause is replaced by a version that is logically equivalent, assuming the rest of E . By $l \not\prec u$ we mean that l is not a variant of u . This allows the larger side of an equation to be simplified by a more general equation, and the smaller side to be rewritten in any case.

As before, we use the notation $E \vdash E'$ to denote one inference step, applying any of the seven rules to a set E of conditional equations to obtain a new set E' . The inference rules are evidently sound, in that the class of provable theorems is unchanged by an inference step.

Let $>$ be any complete simplification ordering extending the given partial ordering \succ . A proof of an equation $s = t$ between ground terms (any variables in s and t may be treated as Skolem constants) is a sequence:

$$s = t_1 \quad \xleftrightarrow{P_1} \quad t_2 \quad \xleftrightarrow{P_2} \quad \cdots \quad \xleftrightarrow{P_m} \quad t_{m+1} = t$$

of $m + 1$ terms ($m \geq 0$), each step $t_k \leftrightarrow t_{k+1}$ of which is either trivial ($t_{k+1} = t_k$), or else is justified by a conditional equation e_k in E , applied at some position in t_k , a substitution σ_k for variables in the equation, and subproofs P_k (of the same form) for each conditions $u_{k,j}\sigma_k = v_{k,j}\sigma_k$ of the applied instance $e_k\sigma_k$. Steps employing an unconditional equation do not have subproofs as part of their justification. By the completeness of positive-unit resolution for Horn clauses, any equation $s = t$ that is valid for a set E of conditional equations is amenable to such an equational proof.

The above inference rules are designed to allow any equational proof to be transformed into normal form. A strategy based on these rules is complete if we can show that, with enough inferences, any theorem has a normal-form proof. For the unit strategy, a *normal-form* proof is a valley proof having no peaks $s \leftarrow u \rightarrow t$, no steps $s \rightarrow t$ with (valley) subproofs, and no trivial steps. Normal-form proofs may be thought of as “direct” proofs; in a refutational framework the existence of such a proof for $s = t$ means that demodulation of s and t using positive unit equations suffices to derive a contradiction between the Skolemized negation $s' \neq t'$ of the given theorem and $x = x$.

We must demonstrate that for any proof $s \leftrightarrow_{E_0}^* t$, there eventually exists an unconditional valley proof $s \downarrow_{E_k} t$. In the unit strategy, only expansions involving an unconditional equation are necessary. Specifically, both equations used by **superpose** are unconditional and the positive literal used in **narrow** is a unit. Were it not for contraction rules, it would be relatively easy to show that **narrow** and **reflect** eventually provide an unconditional proof of $s = t$, and that **superpose** eventually turns that into a valley.

Conditional inference is *fair* if all persistent superpositions of *unit* clauses, narrowings via unit clauses, and reflections have been considered:

9.7. DEFINITION (*Unit Strategy [Dershowitz 1991]*). An inference sequence $E_0 \vdash E_1 \vdash \dots$ is *fair* with respect to the unit strategy if

$$\text{cp}^1(E_\infty) \subseteq E_0 \cup E_1 \cup \dots$$

where E_∞ is the set of *persisting* conditional equations and $\text{cp}^1(E_\infty)$ is the set of conditional equations that may be inferred from persisting equations by one application of **deduce** for unconditional systems, **narrow** with p empty, or **reflect**.

9.8. THEOREM (Dershowitz 1991). *If an inference sequence $E_0 \vdash E_1 \vdash \dots$ is fair for the unit strategy, then for any proof of $s = t$ in E_0 , there is a normal-form proof of $s = t$ in E_∞ .*

PROOF. For the proof ordering, the term ordering $>$ is first extended to the transitive closure of it together with the proper subterm ordering $>_{\text{sub}}$, which is still well-founded [Dershowitz and Jouannaud 1990]. This in turn is extended to equations by considering the equation as a bag of two terms, and using the bag extension of this ordering. An equation is greater than a term iff one of its sides is. Conjunctions of equations are compared as bags of these bags, and a conjunction is larger than a term if one of its conjuncts is. Proofs are measured in the following way: Consider a step $s = w[l\sigma] \leftrightarrow w[r\sigma] = t$ in a ground proof or its subproofs, using a conditional equation $q \Rightarrow l = r$, with $s \geq t$ (in the complete simplification ordering extending $>$). To each such step, we assign the weight $\langle [q\sigma, s, l\sigma], q \Rightarrow l = r \rangle$. Steps are compared in the lexicographic ordering of these pairs. The first components of pairs are compared in the bag extension of the ordering on conjunctions and terms described above. (Note that s is always greater or equal to $l\sigma$, and that for decreasing instances it is also greater than $q\sigma$.) Second components are compared

using the extension \triangleright of the encompassment ordering described earlier. Proofs are compared in the well-founded bag extension of the lexicographic ordering on steps. We use \gg to denote this well-founded proof ordering.

One needs to show that inferences never increase the complexity of proofs and, furthermore, that there are always inferences that can decrease the complexity of nonnormal proofs. Then, by induction with respect to \gg , the eventual existence of a normal-form proof follows: If $E \vdash E'$, then for any proof P in E of an equation $s = t$, there exists a proof P' in E' of $s = t$, such that $P \gg P'$. This is established by consideration of the effects of each contracting inference rule that deletes or replaces equations, since for expansion rules, $E \subseteq E'$, and we can take $P' = P$. The conditions imposed on **compose** are essential for showing a decrease in \gg . Furthermore, if P is a non-normal-form proof in E , then there exists a proof P' in $E \cup \mathbf{cp}^1(E)$ such that $P \gg P'$.

If $s = t$ is provable in E_0 , then it has a proof P in the limit E_∞ . If P is nonnormal, then it admits a smaller proof P' using (in addition to E_∞) a finite number of equations in $\mathbf{cp}^1(E_\infty)$. By fairness, each of those equations appeared at least once along the way. Subsequent inferences can only decrease the complexity of the proof of such an equation once it appears in a set E_i (and has a one-step proof). Thus, each equation needed in P' has a proof of no greater complexity in E_∞ itself, and hence (by the bag nature of the proof measure), there is a proof of $s = t$ in E_∞ that is strictly smaller than P . Since the ordering on proofs is well-founded, by induction there must be a normal proof in E_∞ . \square

In the above method, only unconditional equations are used for superposition and narrowing. This may be good for theorem proving purposes, but is useless from the point of view of completion. An alternative is to design an inference system that distinguishes between decreasing and nondecreasing non-unit clauses [Ganzinger 1991, Dershowitz 1991b]. The required inferences (using **superpose** and **narrow**) are a stringent restriction of paramodulation.

For the decreasing method, we redefine a normal-form proof of $s = t$ to be a valley proof $s \downarrow t$ in which each subproof is also in normal form and each term in a subproof is smaller than the larger of s and t [Dershowitz and Okada 1988]. Any non-normal-form proof has a peak made from decreasing instances with normal-form subproofs, or has a nondecreasing step with normal-form subproofs, or has a trivial step. Theorem 9.5 can be adapted to ground confluence of decreasing systems. Superposition is needed between decreasing conditional rules. As before, we must perform enough expansions with persistent conditional equations for there to always be a normal-form proof in the limit.

9.9. DEFINITION (*Decreasing Strategy* [Dershowitz 1991]). An inference sequence $E_0 \vdash E_1 \vdash \dots$ is *fair* with respect to the *decreasing strategy* if

$$\mathbf{cp}^+(E_\infty) \subseteq E_0 \cup E_1 \cup \dots$$

where $\mathbf{cp}^+(E_\infty)$ is the set of conditional equations that may be inferred from persisting equations by one application of an expansion rule **superpose**, **narrow**, or

reflect.

Decreasingness is essentially the same condition as imposed on conditional rewrite rules by the completion-like procedures of [Kaplan 1987, Ganzinger 1991]. In these methods, superposition is used when the left-hand side is larger than the conditions; narrowing, when a condition dominates the left-hand side. (As theorem provers, however, those were refutationally *incomplete*, since they made no provision for unorientable equations $s = t$ such that $s \not\leq t$ and $t \not\leq s$.)

9.10. THEOREM (Dershowitz 1991). *If an inference sequence is fair for the decreasing strategy, then for any proof of $s = t$ in the initial set E_0 of conditional equations, there is a normal-form proof of $s = t$ in the limit E_∞ .*

For extensions of the ideas in this chapter to full first-order theorem proving, see [Bachmair and Ganzinger 2001] (Chapter 2 of this Handbook).

10. Programming

Rewrite systems are readily used as a programming language. If one requires of the programmer that all programs be terminating, then rewriting may be used as is to compute normal forms. With ground confluence, one is assured of their uniqueness.

Modularity is critical in the programming context. The idea of modularity is to infer properties of a combination of two rewrite systems from properties of their parts:

10.1. THEOREM (Toyama 1987). *The union of two confluent rewrite systems sharing no function symbols or constants is also confluent.*

An example showing that confluence is not preserved when a constructor is shared is:

$$\begin{array}{lcl} f(x, x) & \rightarrow & a \\ f(x, c(x)) & \rightarrow & b \end{array} \quad \left| \quad \begin{array}{lcl} e & \rightarrow & c(e) \end{array}$$

In the combined nonterminating, non-left-linear system, $f(e, e)$ reduces both to a and b .

10.2. THEOREM (Toyama, Klop and Barendregt 1995). *The union of two convergent left-linear rewrite systems sharing no function symbols or constants is also convergent.*

For a proof, see [Marchiori 1995].

These results unfortunately do not carry over to the prevalent situation of shared constructors. One result that does is:

10.3. THEOREM (Gramlich 1995, Dershowitz 1995). *The union of two convergent rewrite systems, sharing only constructor symbols and all of whose critical pairs are overlays, is convergent.*

This is because innermost termination of such systems implies termination, while innermost termination is preserved by such unions [Kurihara and Kaji 1990].

10.4. DEFINITION. We say that two rewrite systems R and S are *mutually-orthogonal* (symbolized $R \perp S$) if there are no non-trivial critical pairs between rules of the different systems.

As a corollary of Theorem 5.15, we have:

10.5. THEOREM. *The union of two mutually-orthogonal rewrite systems is confluent if it is terminating.*

Analogous to Theorem 5.23, we have:

10.6. THEOREM (Raoult and Vuillemin 1980). *The union of two left-linear confluent mutually-orthogonal rewrite systems is confluent.*

The related study of properties of combinations of algebraic rewriting with versions of the lambda calculus began with [Breazu-Tannen and Gallier to appear].

Many programs (interpreters, for example) do not always terminate. Still, we would want to compute normal forms whenever they exist. Confluent systems have at most one normal form per input term, and orthogonal systems are confluent. The left-linearity restriction for orthogonal systems is reasonable in the programming context, since the formal parameters of procedure definitions are distinct. It is also convenient for efficiency of pattern matching. To check if a term $f(s, t)$ is an instance of a left-hand side $f(x, x)$, it is necessary to check that s and t are identical, which can require time proportional to the size of s or t . (Of course, there are also cases where it is very convenient to use non-left-linear rules.)

To find the unique normal form for orthogonal systems, when it exists, one can use the following strategy for choosing the redex at which to apply a rule:

10.7. DEFINITION (*Outermost Rewriting*). A rewriting step $s \rightarrow t$ is *outermost* with respect to some rewrite system if no rule applies at a symbol closer to the root symbol (in the tree representation of terms).

10.8. THEOREM (Outermost Normalization [O'Donnell 1977]). *For any orthogonal system, if no outermost step is perpetually ignored, the normal form—if there is one—will be reached.*

Outermost rewriting of expressions is similarly used to compute normal forms in combinatory logic and head normal forms in the lambda calculus.

In this way, orthogonal systems provide a simple, pattern-directed (first-order) functional programming language, in which the orthogonal conditional operator

$$\begin{aligned} \text{if}(T, x, y) &\rightarrow x \\ \text{if}(F, x, y) &\rightarrow y \end{aligned}$$

can also conveniently be incorporated. Various strategies have been developed for efficient computation in special cases. Moreover, orthogonal systems lend themselves easily to parallel evaluation schemes.

Huet and Lévy [1991] developed a theory of “needed redexes” and optimal derivations for orthogonal systems. The need for a redex is, however, undecidable, except in special cases [Hoffmann and O’Donnell 1982, Huet and Lévy 1991]. Chew [1980] used congruence-closure techniques to cache results of prior sequences of orthogonal rewrites, and improve performance; this idea was extended to a class of non-orthogonal convergent systems in [Verma 1995].

Since programs are often nonterminating, techniques for showing confluence of nonterminating conditional rewrite systems are useful:

10.9. DEFINITION (*Conditional Orthogonality [Bergstra and Klop 1986]*). A conditional rewrite system is *orthogonal* if

1. every variable occurring on the right side or in a condition also appears on the left,
2. each variable occurs at most once in a left-hand side of a rule,
3. one side of each condition is a ground normal form,
4. no left-hand side unifies with a renamed nonvariable subterm of any other left-hand side or with a proper subterm of itself, and
5. no left-hand side is just a variable.

10.10. THEOREM (Bergstra and Klop 1986). *Every orthogonal conditional rewrite system is confluent.*

This definition of orthogonality could be weakened to allow overlaps when the conjunction of the conditions of the overlapping rules cannot be satisfied by the rules of the system. This is the case with the Conditional Append example, since only the last two rules overlap, but $\text{null}(\epsilon)$ can never be F .

As indicated earlier, there are various methods of defining the semantics of conditional rewrite systems. For example, if we have arbitrary conditions as in

$$\begin{array}{l|l} p(c) & a \rightarrow b \\ \neg p(c) & a \rightarrow b \end{array}$$

can we rewrite a to b ? We might say yes, since either $p(c)$ is true or $\neg p(c)$ is. We might say no, since neither condition can be proved. For discussions of logic-based semantics and alternative operational semantics for conditional systems, see [Brand et al. 1979, Plaisted 1987, Dershowitz and Plaisted 1988, Dershowitz and Okada 1990].

Conditional equations provide a natural bridge between functional programming, based on equational semantics, and logic-programming, based on Horn clauses. Note that the above rules can be expressed as

$$\begin{array}{l|l} p(c) = T & a \rightarrow b \\ \neg p(c) = T & a \rightarrow b \end{array}$$

In this way, we can convert conditions involving arbitrary formulæ to conditions involving equations. However, the law of the excluded middle no longer holds; we do not have $x = T$ or $x = F$ for all x . This changes the semantics, of course. Interpreting definite Horn clauses $p \vee \neg q_1 \vee \dots \vee \neg q_n$ as conditional rewrite rules, $q_1 \downarrow T \wedge \dots \wedge q_n \downarrow T \mid p \rightarrow T$, gives a system satisfying the constraints of Theorem 9.3, because predicate symbols are never nested in the “head” p of a clause. Furthermore, all critical pairs are joinable, since all right-hand sides are just T .

However, logic programming permits variables to be bound by unification, whereas conditional rewriting typically uses matching instead, which is more restrictive. To simulate a language like Prolog, something like “conditional narrowing” is needed. See [Dershowitz and Plaisted 1988] for one approach to conditional narrowing. (See [Baader and Snyder 2001, page 495] in Chapter 8 of this Handbook, for the definition of narrowing and related equation-solving methods.) Solving existential queries for conditional equations corresponds to the logic-programming capability of resolution-based languages like Prolog. Goals of the form $s =^? t$ can be solved by a linear restriction of paramodulation akin to narrowing (for unconditional equations) and to the selected linear strategy for Horn-clause logic. If s and t are unifiable, then the goal is satisfied by any instance of their most general unifier. Alternatively, if there is a (renamed) conditional rule $p \mid l \rightarrow r$ such that l unifies with a nonvariable (selected) subterm of s via most general unifier μ , then the conditions in $p\mu$ are solved, say via substitution ρ , and the new goal becomes $s\mu\rho =^? t\mu\rho$.

Suppose we wish to solve

$$\text{append}(x, y) =^? x$$

using Conditional Append (9.1). To apply the conditional rule, we need first to solve $\text{null}(x) =^? F$ using the (renamed) rule $\text{null}(u : v) \rightarrow F$, thereby narrowing the original goal to

$$\text{head}((u : v), \text{append}(\text{tail}(u : v), y)) =^? u : v$$

Straightforward rewriting reduces this to

$$u : \text{append}(v, y) =^? u : v$$

to which the first rule for *append* applies (letting v be ϵ), giving a new goal $u : y =^? u : v$. Since the two terms are now unifiable, this process has produced the solution $x \mapsto u : \epsilon$ and $y, v \mapsto \epsilon$.

For ground confluent conditional systems, any equationally satisfiable goal can be solved by the method outlined above. Some recent proposals for logic programming languages, incorporating equality, adopt such an operational mechanism. The idea of adding rewrite-based equation solving to rewriting to provide a functional-logic language originated with [Dershowitz 1985, Fribourg 1985, Goguen and Meseguer 1986, Dershowitz and Plaisted 1988]. A number of experimental languages combine

narrowing with outermost (“lazy”) evaluation to add goal solving capabilities within functional languages. See [Reddy 1986, Hanus 1994].

Simplification via terminating rules is a very powerful feature, particularly when defined function symbols are allowed to be arbitrarily nested in left-hand sides (which is not permitted with orthogonal rules). Assuming ground convergence, any strategy can be used for simplification, and completeness of the goal-solving process is preserved. One way negation can be handled is by incorporating negative information in the form of rewrite rules which are then used to simplify subgoals to F . Combined with eager simplification, this approach has the advantage of allowing unsatisfiable goals to be pruned, thereby avoiding some potentially infinite paths. Various techniques are also available to help avoid some superfluous paths that cannot lead to solutions.

The semantics of rewriting with infinite structures was explored in [Dershowitz, Kaplan and Plaisted 1991, Kennaway, Klop, Sleep and de Vries 1995].

Acknowledgments

The authors gratefully acknowledge the contributions of Jürgen Giesl, Bernhard Gramlich, Mitch Harris, Konstantin Korovin, Vincent van Oostrum, Albert Rubio, and Rakesh Verma to the preparation of this survey. The National Science Foundation partially supported the writing of this survey under grant NSF CCR-9972118.

Bibliography

- ANANTHARAMAN S. AND HSIANG J. [1990], ‘Automated proofs of the Moufang identities in alternative rings’, *J. Symbolic Computation* **6**, 79–109.
- ARNON D. S., COLLINS G. E. AND MCCALLUM S. [1984], ‘Cylindrical algebraic decomposition. I. The basic algorithm’, *SIAM J. Comput.* **13**(4), 865–877.
- ARTS T. AND GIESL J. [2000], ‘Termination of term rewriting using dependency pairs’, *Theoretical Computer Science* **236**, 133–178.
- AVENHAUS J. AND MADLENER K. [1989], ‘Term rewriting and equational reasoning’. to appear in R.B. Banoji: *Formal Techniques in Artificial Intelligence: A Sourcebook*.
- AVENHAUS J. AND MADLENER K. [1990], Term rewriting and equational reasoning, in R. B. Banerji, ed., ‘Formal Techniques in Artificial Intelligence: A Sourcebook’, Elsevier, Amsterdam, pp. 1–41.
- BAADER F. AND NIKPOT T. [1998], *Term Rewriting and All That*, Cambridge University Press, Cambridge, U.K.
- BAADER F. AND SNYDER W. [2001], Unification theory, in A. Robinson and A. Voronkov, eds, ‘Handbook of Automated Reasoning’, Vol. I, Elsevier Science, chapter 8, pp. 445–532.
- BACHMAIR L. AND DERSHOWITZ N. [1986], Commutation, transformation, and termination, in J. H. Siekmann, ed., ‘Proceedings of the Eighth International Conference on Automated Deduction (Oxford, England)’, Vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 5–20.
- BACHMAIR L. AND DERSHOWITZ N. [1987], Inference rules for rewrite-based first-order theorem proving, in ‘Proceedings of the Second IEEE Symposium on Logic in Computer Science’, Ithaca, NY, pp. 331–337.

- BACHMAIR L. AND DERSHOWITZ N. [1988], 'Critical pair criteria for completion', *J. Symbolic Computation* 6(1), 1-18.
- BACHMAIR L. AND DERSHOWITZ N. [1989], 'Completion for rewriting modulo a congruence', *Theoretical Computer Science* 67(2 & 3).
- BACHMAIR L. AND DERSHOWITZ N. [1994], 'Equational inference, canonical proofs, and proof orderings', *J. of the Association for Computing Machinery* 41(2), 236-276.
- BACHMAIR L., DERSHOWITZ N. AND PLAISTED D. A. [1989], Completion without failure, in H. Aït-Kaci and M. Nivat, eds, 'Resolution of Equations in Algebraic Structures', Vol. 2: Rewriting Techniques, Academic Press, New York, chapter 1, pp. 1-30.
- BACHMAIR L. AND GANZINGER H. [2001], Resolution theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19-99.
- BACHMAIR L. AND PLAISTED D. [1985], 'Termination orderings for associative-commutative rewriting systems', *J. Symbolic Computation* 1, 329-349.
- BAIRD T., PETERSON G. AND WILKERSON R. [1989], Complete sets of reductions modulo associativity, commutativity, and identity, in 'Proceedings of the 3rd International Conference on rewriting techniques and applications', Vol. 355 of *Lecture Notes in Computer Science*, pp. 29-44.
- BALLANTYNE A. M. AND LANKFORD D. S. [1981], 'New decision algorithms for finitely presented commutative semigroups', *J. Computational Mathematics with Applications* 7, 159-165.
- BECKER T. AND WEISPFENNING V. [1993], *Gröbner Bases: A Computational Approach to Commutative Algebra*, Vol. 141 of *Graduate Texts in Mathematics*, Springer-Verlag, Berlin.
- BELLEGARDE F. AND LESCANNE P. [1990], 'Termination by completion', *Applied Algebra on Engineering, Communication and Computer Science* 1(2), 79-96.
- BEN CHERIFA A. AND LESCANNE P. [1987], 'Termination of rewriting systems by polynomial interpretations and its implementation', *Science of Computer Programming* 9(2), 137-159.
- BENNINGHOFFEN B., KEMMERICH S. AND RICHTER M. M. [1987], *Systems of Reductions*, Vol. 277 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.
- BERGSTRA J. A. AND KLOP J. W. [1986], 'Conditional rewrite rules: Confluency and termination', *J. of Computer and System Sciences* 32, 323-362.
- BERTLING H. AND GANZINGER H. [1989], Completion-time optimization of rewrite-time goal solving, in N. Dershowitz, ed., 'Proceedings of the Third International Conference on Rewriting Techniques and Applications (Chapel Hill, NC)', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 45-58.
- BIRKHOFF G. [1935], 'On the structure of abstract algebras', *Proceedings of the Cambridge Philosophical Society* 31, 433-454.
- BOFILL M., GODOY G., NIEUWENHUIS R. AND RUBIO A. [1999], Paramodulation with non-monotonic orderings, in 'Proc. 14th Annual IEEE Symposium on Logic in Computer Science (Trento, Italy)', IEEE Comp. Sci. Press, pp. 225-234.
- BOOK R. V. AND OTTO F. [1993], *String-Rewriting Systems*, Springer-Verlag.
- BORRALLERAS C., FERREIRA M. AND RUBIO A. [2000], Complete monotonic semantic path orderings, in 'Proceedings of the International Conference on Automated Deduction', pp. 346-364.
- BRAND D., DARRINGER J. A. AND JOYNER, W. J. J. [1979], Completeness of conditional reductions, in 'Proceedings of the Fourth Workshop on Automated Deduction', Austin, TX.
- BREAZU-TANNEN V. AND GALLIER J. [to appear], 'Polymorphic rewriting conserves algebraic strong normalization', *Theoretical Computer Science*.
- BROWN, JR. T. C. [1975], A Structured Design-Method for Specialized Proof Procedures, PhD thesis, California Institute of Technology, Pasadena, CA.
- BURRIS S. AND LAWRENCE J. [1991], 'Term rewrite rules for finite fields', *International Journal of Algebra and Computation* 1(3), 353-369.

- BUTLER G. AND LANKFORD D. S. [1980], Experiments with computer implementations of procedures which often derive decision algorithms for the word problem in abstract algebras, Memo MTP-7, Department of Mathematics, Louisiana Tech. University, Ruston, LA.
- CHEW P. [1980], An improved algorithm for computing with equations, in 'Proceedings of the Twenty First Conference on Foundations of Computer Science', Los Angeles, CA, pp. 108-117.
- CHEW P. [1981], Unique normal forms in term rewriting systems with repeated variables, in 'Proceedings of the 13th Annual Symposium on the Theory of Computing', pp. 7-18.
- CHRISTIAN J. [1989], Fast Knuth-Bendix completion: Summary, in N. Dershowitz, ed., 'Proceedings of the Third International Conference on Rewriting Techniques and Applications (Chapel Hill, NC)', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 551-555.
- CICHON A. AND MARION J.-Y. [1999], Light LPO, Rapport de Recherche 99-R-138, Laboratoire Lorrain de Recherche en Informatique et ses Applications.
- CIRSTEA H. AND KIRCHNER C. [1999], Combining higher-order and first-order computation using ρ -calculus: Towards a semantics of ELAN, in 'Frontiers of Combining Systems 2', Studies in Logic and Computation, Research Studies Press, Baldock, England, pp. 95-119.
- COMON H. [1990], Solving inequations in term algebras (Preliminary version), in 'Proceedings of the Fifth Annual Symposium on Logic in Computer Science', IEEE, Philadelphia, PA, pp. 62-69.
- COMON H. [2001], Inductionless induction, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 14, pp. 913-962.
- COMON H., NARENDHAN P., NIEUWENHUIS R. AND RUSINOWITCH M. [1998], Decision problems in ordered rewriting, in 'Proc. of the 13th Annual IEEE Symposium on Logic in Computer Science', Indianapolis, IN, pp. 276-286.
- COMON H. AND TREINEN R. [1997], 'The first-order theory of lexicographic path orderings is undecidable', *Theoretical Computer Science* 176(1-2), 67-87.
- DAUCHET M. [1992], 'Simulation of Turing machines by a regular rewrite rule', *Theoretical Computer Science* 103(2), 409-420.
- DAUCHET M., TISON S., HEULLARD T. AND LESCANNE P. [1987], Decidability of the confluence of ground term rewriting systems, in 'Proceedings of the Second Symposium on Logic in Computer Science', pp. 353-359.
- DESHOWITZ N. [1979], 'A note on simplification orderings', *Information Processing Letters* 9(5), 212-215.
- DESHOWITZ N. [1981], Termination of linear rewriting systems, in 'Proceedings of the Eighth International Colloquium on Automata, Languages and Programming (Acre, Israel)', Vol. 115 of *Lecture Notes in Computer Science*, European Association of Theoretical Computer Science, Springer-Verlag, Berlin, pp. 448-458.
- DESHOWITZ N. [1982], 'Orderings for term-rewriting systems', *Theoretical Computer Science* 17(3), 279-301.
- DESHOWITZ N. [1985], 'Computing with rewrite systems', *Information and Control* 64(2/3), 122-157.
- DESHOWITZ N. [1987], 'Termination of rewriting', *J. Symbolic Computation* 3(1&2), 69-115. Corrigendum: 4, 3 (December 1987), 409-410.
- DESHOWITZ N. [1991a], Canonical sets of Horn clauses, in J. L. Albert, B. Monien and M. R. Artalejo, eds, 'Proceedings of the Eighteenth International Colloquium on Automata, Languages and Programming (Madrid, Spain)', Vol. 510 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 267-278.
- DESHOWITZ N. [1991b], Ordering-based strategies for Horn clauses, in 'Proceedings of the Twelfth International Joint Conference on Artificial Intelligence', Sydney, Australia, pp. 118-124.

- DERSHOWITZ N. [1995], Hierarchical termination, in N. Dershowitz and N. Lindenstrauss, eds, 'Proceedings of the Fourth International Workshop on Conditional and Typed Rewriting Systems (Jerusalem, Israel, July 1994)', Vol. 968 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 89–105.
- DERSHOWITZ N. AND HOOT C. [1995], 'Natural termination', *Theoretical Computer Science* **142**(2), 179–207.
- DERSHOWITZ N., HSIANG J., JOSEPHSON N. AND PLAISTED D. A. [1983], Associative-commutative rewriting, in 'Proceedings of the Eighth International Joint Conference on Artificial Intelligence', pp. 940–944.
- DERSHOWITZ N., HSIANG J. AND SHIENG G.-S. [2000], 'Using Boolean rings and simplification to test satisfiability'. Submitted.
- DERSHOWITZ N. AND JOUANNAUD J.-P. [1990], Rewrite systems, in J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B: Formal Methods and Semantics, North-Holland, Amsterdam, chapter 6, pp. 243–320.
- DERSHOWITZ N., KAPLAN S. AND PLAISTED D. A. [1991], 'Rewrite, rewrite, rewrite, rewrite, rewrite, ...', *Theoretical Computer Science* **83**(1), 71–96.
- DERSHOWITZ N. AND MANNA Z. [1979], 'Proving termination with multiset orderings', *Communications of the ACM* **22**(8), 465–476.
- DERSHOWITZ N., MARCUS L. AND TARLECKI A. [1988], 'Existence, uniqueness, and construction of rewrite systems', *SIAM J. on Computing* **17**(4), 629–639.
- DERSHOWITZ N. AND OKADA M. [1988], Proof-theoretic techniques and the theory of rewriting, in 'Proceedings of the Third IEEE Symposium on Logic in Computer Science', Edinburgh, Scotland, pp. 104–111.
- DERSHOWITZ N. AND OKADA M. [1990], 'A rationale for conditional equational programming', *Theoretical Computer Science* **75**, 111–138.
- DERSHOWITZ N., OKADA M. AND SIVAKUMAR G. [1987], Confluence of conditional rewrite systems, in S. Kaplan and J.-P. Jouannaud, eds, 'Proceedings of the First International Workshop on Conditional Term Rewriting Systems (Orsay, France)', Vol. 308 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 31–44.
- DERSHOWITZ N., OKADA M. AND SIVAKUMAR G. [1988], Canonical conditional rewrite systems, in 'Proceedings of the Ninth Conference on Automated Deduction (Argonne, IL)', Vol. 310 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 538–549.
- DERSHOWITZ N. AND PLAISTED D. A. [1988], Equational programming, in J. E. Hayes, D. Michie and J. Richards, eds, 'Machine Intelligence 11: The logic and acquisition of knowledge', Oxford Press, Oxford, chapter 2, pp. 21–56.
- DERSHOWITZ N. AND REDDY U. [1993], 'Deductive and inductive synthesis of equational programs', *J. Symbolic Computation* **15**, 467–494.
- DETLEFS D. AND FORGAARD R. [1985], A procedure for automatically proving the termination of a set of rewrite rules, in J.-P. Jouannaud, ed., 'Proceedings of the First International Conference on Rewriting Techniques and Applications (Dijon, France)', Vol. 202 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 255–270.
- DEVIE H. [1990], When ordered completion fails, in M. Okada, ed., 'Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems (Montreal, Canada)', Vol. 516 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.
- DICK J., KALMUS J. AND MARTIN U. [1990], 'Automating the Knuth Bendix ordering', *Acta Informatica* **28**(2), 95–119.
- DOWES G., HARDIN T. AND KIRCHNER C. [1994], 'Higher-order unification via explicit substitutions'.
- EDELSON R. [1990]. Private Communication.
- EVANS T. [1951], 'On multiplicative systems defined by generators and relations, I', *Proceedings of the Cambridge Philosophical Society* **47**, 637–649.

- FERREIRA M. C. F. AND ZANTEM H. [1993], Total termination of term rewriting, in C. Kirchner, ed., 'Proceedings of the Fifth Conference on Rewriting Techniques and Applications', Vol. 690 of *Lecture Notes in Computer Science*, Springer, pp. 213–227.
- FORET A. [1988], Rewrite rule systems for modal propositional logic, in 'Proceedings of an International Workshop on Algebraic and Logic Programming', Akademie-Verlag, Gausig, GDR, pp. 146–157.
- FRIBOURG L. [1985], SLOG: A logic programming language interpreter based on clausal superposition and rewriting, in 'Proceedings of the Symposium on Logic Programming', IEEE, Boston, MA, pp. 172–184.
- GALLIER J. [1991], 'What's so special about Kruskal's Theorem and the ordinal Γ_0 . A survey of some results in proof theory', *Annals of Pure and Applied Logic* **53**(3), 199–260.
- GANZINGER H. [1991], 'A completion procedure for conditional equations', *Journal of Symbolic Computation* **11**, 51–81.
- GENET T. AND GNAEDIG I. [1997], Termination proofs using GPO ordering constraints, in M. Dauchet, ed., 'Proceedings 22nd International Colloquium on Trees in Algebra and Programming (Lille, France)', Vol. 1214 of *Lecture Notes in Computer Science*, pp. 249–260.
- GEUPEL O. [1989], Overlap closures and termination of term rewriting systems, Report MIP-8922, Universität Passau, Passau, West Germany.
- GIESL J. [1995], Generating polynomial orderings for termination proofs, in 'Proceedings of the Sixth International Conference on Rewriting Techniques and Applications (Kaiserslautern, Germany)', Vol. 914 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 426–431.
- GIOVANNETTI E. AND MOISO C. [1987], A completeness result for conditional narrowing, in 'Presented at the First International Workshop on Conditional Term Rewriting Systems', Orsay, France.
- GOGUEN J. A., KIRCHNER C. AND MESEGUER J. [1987], Concurrent term rewriting as a model of computation, in R. Keller and J. Fasel, eds, 'Proceedings of Graph Reduction Workshop (Santa Fe, NM)', Vol. 279 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 53–93.
- GOGUEN J. A. AND MESEGUER J. [1986], EQLOG: Equality, types, and generic modules for logic programming, in D. DeGroot and G. Lindstrom, eds, 'Logic Programming: Functions, Relations, and Equations', Prentice-Hall, Englewood Cliffs, NJ, pp. 295–363.
- GRAMLICH B. [1995], 'Abstract relations between restricted termination and confluence properties of rewrite systems', *Fundamenta Informaticae* **24**, 3–23.
- GRAMLICH B. AND WIRTH C.-P. [1996], Confluence of terminating conditional rewrite systems revisited, in H. Ganzinger, ed., 'Proceedings of the International Conference on Rewriting Techniques and Applications', Vol. 1103 of *Lecture Notes in Computer Science*, Springer-Verlag, New Brunswick, NJ, USA, pp. 245–259.
- GUTTAG J. V., KAPUR D. AND MUSSER D. R. [1983], 'On proving uniform termination and restricted termination of rewriting systems', *SIAM J. on Computing* **12**(1), 189–214.
- HANUS M. [1994], 'The integration of functions into logic programming: From theory to practice', *J. Logic Programming* **19&20**, 583–628.
- HANUS M. [1995], On extra variables in (equational) logic programming, in 'Proc. Twelfth International Conference on Logic Programming', MIT Press, pp. 665–679.
- HIGMAN G. [1952], 'Ordering by divisibility in abstract algebras', *Proceedings of the London Mathematical Society* (3) **2**(7), 326–336.
- HIGMAN G. AND NEUMANN B. [1952], 'Groups as groupoids with one law', *Publ. Math. Debrecen* **2**, 215–221.
- HINDLEY J. R. [1964], The Church-Rosser Property and a Result in Combinatory Logic, PhD thesis, University of Newcastle-upon-Tyne.
- HOFBAUER D. [1992], 'Termination proofs by multiset path orderings imply primitive recursive derivation lengths', *Theoretical Computer Science* **105**(1), 129–140.

- HOFBAUER D. AND LAUTEMANN C. [1989], Termination proofs and the length of derivations (preliminary version), in 'Proceedings of the 3rd International Conference on rewriting techniques and applications', Vol. 355 of *Lecture Notes in Computer Science*, pp. 167-177.
- HOFFMANN C. M. AND O'DONNELL M. J. [1982], 'Programming with equations', *ACM Transactions on Programming Languages and Systems* 4(1), 83-112.
- HSIANG J. [1985], 'Refutational theorem proving using term-rewriting systems', *Artificial Intelligence* 25, 255-300.
- HSIANG J. AND DERSHOWITZ N. [1983], Rewrite methods for clausal and non-clausal theorem proving, in 'Proceedings of the Tenth International Colloquium on Automata, Languages and Programming (Barcelona, Spain)', Vol. 154 of *Lecture Notes in Computer Science*, European Association of Theoretical Computer Science, Springer-Verlag, Berlin, pp. 331-346.
- HSIANG J. AND HUANG G. S. [1996], Some fundamental properties of Boolean ring normal forms, in D. Du, J. Gu and P. M. Pardalos, eds, 'Satisfiability Problem: Theory and Applications', Vol. 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, NSF Science and Technology Center, American Mathematical Society, pp. 587-602.
- HSIANG J. AND RUSINOWITCH M. [1987], On word problems in equational theories, in T. Ottmann, ed., 'Proceedings of the Fourteenth EATCS International Conference on Automata, Languages and Programming (Karlsruhe, West Germany)', Vol. 267 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 54-71.
- HSIANG J. AND RUSINOWITCH M. [1991], 'Proving refutational completeness of theorem proving strategies. The transfinite semantic tree method', *J. of the Association for Computing Machinery* 38(3), 559-587.
- HUET G. [1980], 'Confluent reductions: Abstract properties and applications to term rewriting systems', *J. of the Association for Computing Machinery* 27(4), 797-821.
- HUET G. [1981], 'A complete proof of correctness of the Knuth-Bendix completion algorithm', *J. Computer and System Sciences* 23(1), 11-21.
- HUET G. [1985], Cartesian closed categories and Lambda-calculus, in 'Proceedings of the LITP Spring School on Combinators and Functional Programming Languages', Vol. 242 of *Lecture Notes in Computer Science*, Springer-Verlag, Val d'Ajol, France, pp. 123-135.
- HUET G. AND LANKFORD D. S. [1978], On the uniform halting problem for term rewriting systems, Rapport laboria 283, Institut de Recherche en Informatique et en Automatique, Le Chesnay, France.
- HUET G. AND LÉVY J.-J. [1991], Computations in orthogonal rewriting systems, I and II, in J.-L. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of Alan Robinson', MIT Press, Cambridge, MA, pp. 395-443.
- HULLOT J.-M. [1980], A catalogue of canonical term rewriting systems, Technical Report CSL-113, SRI International, Menlo Park, CA.
- JOUANNAUD J.-P. AND KIRCHNER H. [1986], 'Completion of a set of rules modulo a set of equations', *SIAM J. on Computing* 15, 1155-1194.
- JOUANNAUD J.-P. AND LESCANNE P. [1982], 'On multiset orderings', *Information Processing Letters* 15, 57-63.
- JOUANNAUD J.-P., LESCANNE P. AND REINIG F. [1982], Recursive decomposition ordering, in D. Bjørner, ed., 'Proceedings of the Second IFIP Workshop on Formal Description of Programming Concepts', North-Holland, Garmisch-Partenkirchen, West Germany, pp. 331-348.
- JOUANNAUD J.-P. AND OKADA M. [1991], Satisfiability of systems of ordinal notations with the subterm property is decidable, in J. L. Albert, B. Monien and M. R. Artalejo, eds, 'Proceedings of the Eighteenth EATCS Colloquium on Automata, Languages and Programming (Madrid, Spain)', Vol. 510 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 455-468.
- KAMIN S. AND LÉVY J.-J. [1980], Two generalizations of the recursive path ordering, Unpublished note, Department of Computer Science, University of Illinois, Urbana, IL.
- KAPLAN S. [1987], 'Simplifying conditional term rewriting systems: Unification, termination and confluence', *J. Symbolic Computation* 4(3), 295-334.

- KAPLAN S. [1988], 'Rewriting with a nondeterministic choice operator', *Theoretical Computer Science* **56**, 37–57.
- KAPUR D., MUSSER D. AND NARENDHAN P. [1988], 'Only prime superpositions need be considered in the Knuth-Bendix procedure', *Journal of Symbolic Computation* **6**(1), 19–36.
- KAPUR D. AND NARENDHAN P. [1985a], An equational approach to theorem proving in first-order predicate calculus, in 'Proceedings of the Ninth International Joint Conference on Artificial Intelligence', Los Angeles, CA, pp. 1146–1153.
- KAPUR D. AND NARENDHAN P. [1985b], 'A finite Thue system with decidable word problem and without equivalent finite canonical system', *Theoretical Computer Science* **35**, 337–344.
- KAPUR D., NARENDHAN P. AND OTTO F. [1990], 'On ground confluence of term rewriting systems', *Information and Computation* **86**(1), 14–31.
- KAPUR D. AND SIVAKUMAR G. [1997], A total, ground path ordering for proving termination of AC-rewrite systems, in 'Proceedings of the Eighth International Conference on Rewriting Techniques and Applications (Barcelona, Spain)', Vol. 1232 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 142–156.
- KAPUR D. AND ZHANG H. [1989], A case study of the completion procedure: Proving ring commutativity problems. Unpublished Draft.
- KENNAWAY R., KLOP J. W., SLEEP R. AND DE VRIES F.-J. [1995], 'Transfinite reductions in orthogonal term rewriting systems', *Information and Computation* **119**(1), 18–38.
- KIRCHNER C., KIRCHNER H. AND RUSINOWITCH M. [1990], 'Deduction with symbolic constraints', *RAIRO Theoretical Informatics and Applications* **4**(3), 9–52. Special issue on Automatic Deduction.
- KIRCHNER H. AND HERMANN M. [1990], Meta-rule synthesis from crossed rewrite systems, in '2nd International Workshop on Conditional and Typed Rewriting Systems (Montreal, Canada)', Vol. 516 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 143–154.
- KLOP J. W. [1980], *Combinatory Reduction Systems*, Centre for Mathematics and Computer Science, Amsterdam. Mathematical Centre Tracts No. 127.
- KLOP J. W. [1992], Term rewriting systems, in S. Abramsky, D. M. Gabbay and T. S. E. Maibaum, eds, 'Handbook of Logic in Computer Science', Vol. 2, Oxford University Press, Oxford, chapter 1, pp. 1–117.
- KNUTH D. E. AND BENDIX P. B. [1970], Simple word problems in universal algebras, in J. Leech, ed., 'Computational Problems in Abstract Algebra', Pergamon Press, Oxford, U. K., pp. 263–297. Reprinted in *Automation of Reasoning 2*, Springer-Verlag, Berlin, pp. 342–376 (1983).
- KOROVIN K. AND VORONKOV A. [2000], Knuth-Bendix constraint solving is NP-complete, Preprint CSPP-8, Department of Computer Science, University of Manchester.
- KOROVIN K. AND VORONKOV A. [2001], Verifying orientability of rewrite rules using the Knuth-Bendix order, in 'Proceedings of the Twelfth International Conference on Rewriting Techniques and Applications', Lecture Notes in Computer Science, Springer-Verlag.
- KOUNALIS E. AND RUSINOWITCH M. [1988], On word problems in Horn theories, in E. Lusk and R. Overbeek, eds, 'Proceedings of the Ninth International Conference on Automated Deduction (Argonne, Illinois)', Vol. 310 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 527–537.
- KRISHNAMOORTHY M. S. AND NARENDHAN P. [1985], 'On recursive path ordering', *Theoretical Computer Science* **40**, 323–328.
- KRUSKAL J. [1960], 'Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture', *Transactions of the American Mathematical Society* **95**, 210–225.
- KURIHARA M. AND KAJI I. [1990], 'Modular term rewriting systems and the termination', *Information Processing Letters* **34**, 1–4.
- LANKFORD D., BUTLER G. AND BALLANTYNE A. [1984], A progress report on new decision algorithms for finitely presented Abelian groups, in R. E. Shostak, ed., 'Proceedings of the Seventh International Conference on Automated Deduction (Napa, CA)', Vol. 170 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 128–141.

- LANKFORD D. S. [1975], Canonical inference, Memo ATP-32, Automatic Theorem Proving Project, University of Texas, Austin, TX.
- LANKFORD D. S. [1977], Some approaches to equality for computational logic: A survey and assessment, Memo ATP-36, Automatic Theorem Proving Project, University of Texas, Austin, TX.
- LANKFORD D. S. [1979], On proving term rewriting systems are Noetherian, Memo MTP-3, Mathematics Department, Louisiana Tech. University, Ruston, LA. Revised October 1979.
- LANKFORD D. S. [1980], The uniform word problem for J -algebras is decidable, Memo MTP-10, Department of Mathematics, Louisiana Tech. University, Ruston, LA.
- LANKFORD D. S. AND BALLANTYNE A. M. [1977], Decision procedures for simple equational theories with commutative-associative axioms: Complete sets of commutative-associative reductions, Memo ATP-39, Department of Mathematics and Computer Sciences, University of Texas, Austin, TX.
- LE CHENADEC P. [1985], *Canonical Forms in Finitely Presented Algebras*, Pitman-Wiley, London.
- LESCANNE P. [1984], Term rewriting systems and algebra, in R. E. Shostak, ed., 'Proceedings of the Seventh International Conference on Automated Deduction (Napa, CA)', Vol. 170 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 166–174.
- LESCANNE P. [1990], 'On the recursive decomposition ordering with lexicographical status and other related orderings', *J. Automated Reasoning* 6, 39–49.
- MANNA Z. AND NESS S. [1970], On the termination of Markov algorithms, in 'Proceedings of the Third Hawaii International Conference on System Science', Honolulu, HI, pp. 789–792.
- MARCHIORI M. [1995], Modularity of completeness revisited, in 'Proceedings of the Sixth International Conference on Rewriting Techniques and Applications (Kaiserslautern, Germany)', Vol. 914 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 2–10.
- MARCHIORI M. [1996], Unravelings and ultra-properties, in 'Proc. Algebraic and Logic Programming', Vol. 1139 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 107–121.
- MARTIN U. [1989], 'A geometrical approach to multiset orderings', *Information Processing Letters* 67, 37–54.
- MARTIN U. AND NIPKOW T. [1990], Ordered completion, in M. Stickel, ed., 'Proceedings of the Tenth International Conference on Automated Deduction (Kaiserslautern, West Germany)', Vol. 449 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 366–380.
- MCCUNE W. [1997], 'Solution of the Robbins problem', *J. Automated Reasoning* 19(3), 263–276.
- MCCUNE W. W. [1989], *Otter 1.0 Users' Guide*, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois.
- MÉTIVIER Y. [1983], 'About the rewriting systems produced by the Knuth-Bendix completion algorithm', *Information Processing Letters* 16(1), 31–34.
- MUSSER D. [1980], On proving inductive properties of abstract data types, in 'Proceedings, 7th ACM Symposium on Principles of Programming Languages', pp. 154–162.
- NAOI T. AND INAGAKI Y. [1989], Algebraic semantics and complexity of term rewriting systems, in 'Proceedings of the 3rd International Conference on rewriting techniques and applications', Vol. 355 of *Lecture Notes in Computer Science*, pp. 311–325.
- NARENDRA P., RUSINOWITCH M. AND VERMA R. [1998], RPO constraint solving is in NP, in 'Proceedings of the Annual Conference of the European Association of Computer Science Logic (Brno, Czech Republic)', Vol. 1584 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 385–398.
- NASH-WILLIAMS C. S. J. A. [1963], 'On well-quasi-ordering finite trees', *Proceedings of the Cambridge Philosophical Society* 59(4), 833–835.
- NELSON C. G. AND OPPEN D. C. [1980], 'Fast decision procedures based on congruence closure', *J. of the Association for Computing Machinery* 27(2), 356–364.
- NEWMAN M. H. A. [1942], 'On theories with a combinatorial definition of 'equivalence'', *Annals of Mathematics* 43(2), 223–243.

- NIEUWENHUIS R. [1993], 'Simple LPO constraint solving methods', *Information Processing Letters* 47(2).
- NIEUWENHUIS R. AND RUBIO A. [2001], Paramodulation-based theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 7, pp. 371–443.
- NIPKOW T. [1990], 'Unification in primal algebras, their powers and their varieties', *J. of the Association for Computing Machinery* 37, 742–776.
- O'DONNELL M. J. [1977], *Computing in systems described by equations*, Vol. 58 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.
- OHLEBUSCH E. [1999], On quasi-reductive and quasi-simplifying deterministic conditional rewrite systems, in 'Proceedings of the 4th International Symposium on Functional and Logic Programming', Vol. 1722 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 179–193.
- OYAMAGUCHI M. [1987], 'The Church-Rosser property for ground term rewriting systems is decidable', *Theoretical Computer Science* 49(1), 43–79.
- PEDERSEN J. [1984a], Confluence Methods and the Word Problem in Universal Algebra, PhD thesis, Emory University.
- PEDERSEN J. [1985a], Obtaining complete sets of reductions and equations without using special unification algorithms, in B. F. Caviness, ed., 'Proceedings of the European Conference on Computer Algebra: Research Contributions (Linz, Austria)', Vol. 204 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 422–423.
- PEDERSEN J. [1985b], 'The word problem in absorbing varieties', *Houston Journal of Mathematics* 11(4), 575–590.
- PEDERSEN J. [1988], Computer solution of word problems in universal algebra, in M. Tangora, ed., 'Computers in Algebra', Vol. 111 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 103–128.
- PEDERSEN J. [1989], Morphocompletion for one-relation monoids, in N. Dershowitz, ed., 'Proceedings of the Third International Conference on Rewriting Techniques and Applications (Chapel Hill, NC)', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 574–578.
- PEDERSEN J. F. [1984b], Confluence methods and the word problem in universal algebra, PhD thesis, Emory University.
- PETERSON G. [1990], Complete sets of reductions with constraints, in M. Stickel, ed., 'Proceedings of the 10th International Conference on Automated Deduction', pp. 381–395.
- PETERSON G. E. AND STICKEL M. E. [1981], 'Complete sets of reductions for some equational theories', *J. of the Association for Computing Machinery* 28(2), 233–264.
- PLAISTED D. [1978], A recursively defined ordering for proving termination of term rewriting systems, technical report R-78-943, University of Illinois at Urbana-Champaign, Urbana, IL.
- PLAISTED D. [1986], A simple non-termination test for the Knuth-Bendix method, in 'Proceedings of the Eighth International Conference on Automated Deduction', Vol. 230 of *Lecture Notes in Computer Science*, pp. 79–88.
- PLAISTED D. [1987], A logic for conditional term rewriting systems, in S. Kaplan and J.-P. Jouannaud, eds, 'Proceedings of the First International Workshop on Conditional Term Rewriting Systems (Orsay, France)', Vol. 308 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 212–227.
- PLAISTED D. A. [1985], 'The undecidability of self embedding for term rewriting systems', *Information Processing Letters* 20(2), 61–64.
- PLAISTED D. A. [1993], Equational reasoning and term rewriting systems, in D. Gabbay, C. Hogger, J. A. Robinson and J. Siekmann, eds, 'Handbook of Logic in Artificial Intelligence and Logic Programming', Vol. 1, Oxford University Press, Oxford, chapter 5, pp. 273–364.
- PORAT S. AND FRANCEZ N. [1985], Fairness in term rewriting systems, in J.-P. Jouannaud, ed., 'Proceedings of the First International Conference on Rewriting Techniques and Applications

- (Dijon, France)', Vol. 202 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 287–300.
- RAOULT J.-C. AND VUILLEMIN J. [1980], 'Operational and semantic equivalence between recursive programs', *J. of the Association for Computing Machinery* **27**(4), 772–796.
- REDDY U. S. [1986], On the relationship between logic and functional languages, in D. DeGroot and G. Lindstrom, eds, 'Logic Programming: Functions, Relations, and Equations', Prentice-Hall, Englewood Cliffs, NJ, pp. 3–36.
- RENEGAR J. [1992], 'On the computational complexity and geometry of the first-order theory of the reals. III. Quantifier elimination', *J. Symbolic Computation* **13**(3), 329–352.
- ROSEN B. K. [1973], 'Tree-manipulating systems and Church-Rosser theorems', *J. of the Association for Computing Machinery* **20**(1), 160–187.
- RUBIO A. [1999], A fully syntactic AC-RPO, in P. Narendran and M. Rusinowitch, eds, 'Proceedings of the International Conference on Rewriting Techniques and Applications', Vol. 1631 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 133–147.
- RUBIO A. AND NIEUWENHUIS R. [1993], A precedence-based total AC-compatible ordering, in C. Kirchner, ed., 'Proceedings of the Fifth International Conference on Rewriting Techniques and Applications (Montreal, Canada)', Vol. 690 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.
- RUSINOWITCH M. [1987], 'Path of subterms ordering and recursive decomposition ordering revisited', *J. Symbolic Computation* **3**(1&2), 117–132.
- SIEKMANN J. AND SZABO P. [1982], 'A Noetherian and confluent rewrite system for idempotent semigroups', *Semigroup Forum* **25**(1/2), 83–110.
- SMULLYAN R. [1985], *To Mock a Mockingbird*, Knopf.
- SNYDER W. [1989], Efficient ground completion: An $O(n \log n)$ algorithm for generating reduced sets of ground rewrite rules equivalent to a set of ground equations e , in N. Dershowitz, ed., 'Proceedings of the Third International Conference on Rewriting Techniques and Applications (Chapel Hill, NC)', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 419–433.
- STEINBACH J. [1994], 'Generating polynomial orderings', *Information Processing Letters* **49**(2), 85–93.
- STONE M. [1936], 'The theory of representations for Boolean algebra', *Transactions of the American Mathematical Society* **40**, 37–111.
- TARSKI A. [1951], *A Decision Method for Elementary Algebra and Geometry*, University of California Press, Berkeley, CA.
- THOMAS W. [1990], Automata on infinite objects, in J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B: Formal Methods and Semantics, North-Holland, Amsterdam, chapter 4, pp. 133–191.
- THUE A. [1914], 'Probleme über veränderungen von zeichenreihen nach gegeben regeln', *Skr. Vid. Kristiania I. Mat. Naturv. Klasse* **10**/34.
- TISON S. [1989], Fair termination is decidable for ground systems, in 'Proceedings of the Third International Conference on Rewriting Techniques and Applications', Vol. 355 of *Lecture Notes in Computer Science*, pp. 462–476.
- TOYAMA Y. [1987], 'On the Church-Rosser property for the direct sum of term rewriting systems', *J. of the Association for Computing Machinery* **34**(1), 128–143.
- TOYAMA Y., KLOP J. W. AND BARENDREGT H. P. [1995], 'Termination for direct sums of left-linear complete term rewriting systems', *J. of the Association for Computing Machinery* **42**(6), 1275–1304.
- VAN OOSTROM V. [1994], 'Confluence by decreasing diagrams', *Theoretical Computer Science* **126**, 259–280.
- VERMA R. M. [1995], 'A theory of using history for equational systems with applications', *J. of the Association for Computing Machinery* **42**(5), 984–1020.

- VERMA R. M., RUSINOWITCH M. AND LUGIEZ D. [2001], 'Algorithms and reductions for rewriting systems', *Fundamenta Informaticae* p. to appear.
- WEIERMANN A. [1995], 'Termination proofs via the lexicographic path ordering yields multiply recursive derivation lengths', *Theoretical Computer Science* **139**, 355–362.
- WINKLER F. AND BUCHBERGER B. [1983], A criterion for eliminating unnecessary reductions in the Knuth-Bendix algorithm, in 'Proceedings of the Colloquium on Algebra, Combinatorics and Logic in Computer Science'.
- YASUHARA A. [1971], *Recursive Function Theory and Logic*, Academic Press.
- ZANTEMA H. [1994], 'Termination of term rewriting: Interpretation and type elimination', *J. Symbolic Computation* **17**, 23–50.
- ZANTEMA H. [1995], 'Termination of term rewriting by semantic labelling', *Fundamenta Informaticae* **24**, 89–105.
- ZHANG H. [1994], 'A new method for the Boolean ring based theorem proving', *J. Symbolic Computation* **17**(2), 189–211.
- ZHANG H. AND KAPUR D. [1990], 'Unnecessary inferences in associative-commutative completion procedures', *Mathematical Systems Theory* **23**, 175–206.
- ZHEGALKIN I. I. [1927], 'On a technique of evaluation of propositions in symbolic logic', *Matematicheskii Sbornik* **34**(1), 9–27.

Index

A

Abelian groups572, 579, 580, 583
 absorbing rule583
 Ackermann's function553
 append586, 596
 arity541
 associative-commutative rewriting577

B

bag548, 582
 bag extension548
 bimodules580
 Boolean rings584

C

canonical system543
 Cartesian closed categories566
 Chameleon Island538, 546, 579
 Church-Rosser modulo576
 Church-Rosser property 543, 559, 560, 576
 class confluence576
 class rewriting575
 closure542
 combinatory logic565, 566, 594
 commutative axiom574
 commutative rings580
 commute561
 complete simplification ordering571
 completeness543
 completion543, 567, 570, 588
 conditional589
 constrained582
 ordered545
 composition542
 conditional completion589
 conditional critical pair587
 conditional equation586
 conditional rewrite systems586
 conditional rewriting585
 confluence543, 559-561
 ground560
 local562
 strong561
 weak562
 congruence closure568
 consequence
 logical543
 constraints557
 constructor542
 context541

contract542
 convergence543
 convergent system561
 convertible560
 Coxeter groups583
 critical pair562, 563
 conditional587
 ordered569
 Critical Pair Lemma563

D

decision procedure
 equational561
 decision procedures561
 decreasing strategy592
 decreasing system588
 deduction modulo580
 dependency pairs558
 derivation543
 derivation length547
 derives560
 Diamond Lemma562
 distributivity554

E

embedding
 homeomorphic549
 encompassment564
 equation542
 equational system542
 equivalence classes575
 extended rewriting575
 extensions577

F

factorial555, 566
 fair termination547
 fairness571, 591
 fields583
 flattening577
 Fortran567
 forward closures558

G

Gödel585
 Grecian urn538, 546, 580
 Gröbner basis581
 ground confluence560
 ground instance542
 ground systems560

group theory	563, 574
groupoid	568
groups	583
Abelian	572, 579, 580, 583

H

Hercules	555
Hilbert's Tenth Problem	576
Hindley-Rosen Lemma	561
hole	541
homeomorphic embedding	549
Horn clause, equational	586
Hydra	537

I

incrementality	556
induction	560
inductionless induction	585
innermost	542
insertion sort	538, 554
instance	542
interpreter	539, 546, 565
invariance	
full	546
irreducible	543

J

J-algebras	583
join systems	586
joinable	560

K

Knuth-Bendix ordering	557
König's lemma	547, 552

L

lambda calculus	565, 594
lattices	581, 583
distributive	580
left linearity	542, 594
lexicographic path ordering	553
linear	541
local confluence	562
logics	
rewriting	544
loops	539, 545, 567, 583

M

maximal ordering	557
model	543
modularity	541, 593
modules	580
monotonicity	546
most general unifier	562

Moufang identities	583
multiset	548
multiset path ordering	554

N

Newman's Lemma	562
Noetherian	587
non-erasing systems	566
nondeterminism	567
normal form	543, 544
normal form proof	591
normalization	
unique	545
normalizing	545
numeric path ordering	557

O

ordered rewriting	542
ordered simplification	582
ordering	
incremental	556
Knuth-Bendix	557
lexicographic path	553
multiset path	554
numeric path	557
precedence	553
quasi-simplification	549
recursive decomposition	556
recursive path	553
reduction	547
simplification	549
stable	546
subterm	547
well-founded	546
ordinal notations	556
ordinals	556
orthogonality	565, 566, 594, 595
outermost	542
outermost normalization	594
outermost rewriting	594
overlap closures	558
overlapping	563
overlay	563, 587, 588

P

paramodulation	559
partial ordering	541
positions	541
precedence	553
primitive recursive function	555
program synthesis	574
Prolog	586, 596

Q

quasi-ordering	541, 547
quasi-simplification ordering	549
quasi-varieties	588

R

recursive decomposition ordering	556
recursive function	561
recursive path ordering	553, 556
redex	542
reduced convergent system	564
reduced system	564
reducible	543
reduction ordering	547
redundancy	573
relativized rewriting	574
relativized termination	580
rewrite proof	544, 560
rewrite relation	542, 546
rewrite rule	541, 542
rewrite system	542
completeness	543
conditional	586
convergent	543
ground	560, 568
higher-order	541
orthogonal	539
string	540
rewrite systems	
conditional	586
rewriting	542
conditional	585
relativized	574
right linearity	542
rings	580, 583
commutative	580
Robbins algebra	583

S

satisfiability	543
self-embedding	550, 552
semantic matching	576
semantic unification	576
semantics	543, 558, 595
semi-equational systems	586
semi-Thue systems	540
sequence ordering	547
sequentiality	541
simplification ordering	549
strict	549
Skolemization	568, 584, 590, 591
stable extension	546
stack	540

standard systems	586
status	556
string rewriting	540
strong confluence	561
strong normalization	543
substitution	542
ground	542
subterm	541
subterm property	571
subterms.	
disjoint	542
superposition	544

T

term	541
ground	541
term-rewriting system	542
termination	543, 546, 573
fair	547
relativized	580
simple	550
weak	545
termination function	551
theory	
equational	543
total recursive function	556
Tree Theorem	550

U

unifiable	562
unifier	562
unit strategy	591

V

validity	543, 544
valley proof	544
variant	542
variety	545
varyadic	541

W

weak confluence	562
well-foundedness	546
well-ordering	546
Whitman normal form	583
word problem	545
universal	545

Z

Zorn's Lemma	546
--------------	-----

Equality Reasoning in Sequent-Based Calculi

Anatoli Degtyarev

Andrei Voronkov

SECOND READERS: Nikolai Björner.

Contents

1	Introduction	613
1.1	Some useful notation	614
1.2	Equality. The first axiomatization	615
1.3	Paramodulation and its refinements	616
1.4	Equality in sequent systems	622
1.5	Overview of calculi introduced in this chapter	628
2	Translation of logic with equality into logic without equality	628
2.1	Modification method	630
2.2	Constrained equality elimination	633
3	Free variable systems	637
3.1	Free variable tableaux	637
3.2	Matings	640
3.3	The inverse method	641
4	Early history	644
5	Simultaneous rigid E -unification	646
5.1	Definition and examples	646
5.2	Undecidability and other properties of simultaneous rigid E -unification	648
5.3	Other works	652
6	Incomplete procedures for rigid E -unification	653
6.1	Gallier et.al.'s procedure: merits and problems	653
6.2	An incomplete calculus for rigid E -unification	654
6.3	Answer constraints and the tableau method	657
7	Sequent-based calculi and paramodulation	660
7.1	Fitting's tableau paramodulation	660
7.2	Tableau basic superposition	663
7.3	Rigid variables in resolution theorem proving	664
7.4	From resolution to tableaux and matings	667
8	Equality elimination	667
8.1	Theoretical basis for equality elimination	668
8.2	Equality elimination for semantic tableaux	674

8.3	Equality elimination for the inverse method	676
8.4	Further issues in equality elimination	678
9	Equality reasoning in nonclassical logics	679
9.1	Intuitionistic logic with equality	679
9.2	Simultaneous rigid E -unification is inevitable	683
9.3	Automated reasoning modulo simultaneous rigid E -unification	684
9.4	Modal logics with equality	690
10	Conclusion and open problems	691
10.1	Simultaneous rigid E -unification	692
10.2	Equality reasoning for model elimination and other tableau-based methods . .	692
10.3	Equality reasoning in nonclassical logics	692
	Bibliography	693
	Calculi and inference rules	703
	Index	704

1. Introduction

Handling equality in resolution theorem proving is one of the central topics of research in automated reasoning. The first general method based on paramodulation was proposed already by Robinson and Wos [1969] (although there were some earlier publications based on other techniques). The treatment of equality in sequent-based machine-oriented calculi has much longer history started by Wang [1960] and Kanger [1963]. Ideas of Kanger related to equality handling were more thoroughly expressed by Lifschitz [1968] and Orevkov [1969].

The next generation of works in this area is based on rigid unification introduced by Gallier, Raatz and Snyder [1987]. Recently, this area witnessed a rapid development of new results and techniques, including results on rigid E -unification, the equality elimination method, rigid superposition and rigid paramodulation. It has also been discovered that rigid unification has close connections with intuitionistic logic with equality, second-order unification and some combinatorial problems.

In this chapter we describe automated reasoning techniques for all main sequent-based methods of automated deduction, including the tableau method, the connection method, model elimination and the inverse method. We illustrate main results and ideas on the tableau method and sometimes on the inverse method (as a typical backward and a typical forward proof search methods in sequent calculi).

The chapter covers the following topics:

- methods of proof-search in sequent calculi;
- early history of sequent-based automated deduction;
- translation of logic with equality into logic without equality;
- theorem proving using simultaneous rigid E -unification;
- tableau calculi with rigid paramodulation/superposition;
- the equality elimination method; and
- equality reasoning in nonclassical logics.

Due to the rapidly growing interest in tableau-based theorem proving, and because the usual techniques of handling equality in automated reasoning do not straightforwardly generalize to tableaux, we believe that equality reasoning in tableaux and related procedures will be among the central topics in automated deduction in the nearest future. We present many new results that have not yet been presented in any systematic way. We do not assume that readers have any special knowledge of equality reasoning.

In this section we introduce the reader in the area of equality reasoning. In Section 1.1 we introduce the main notation used in the chapter. In Section 1.2 we discuss the standard axiomatization of equality. In Section 1.2 we consider several techniques of handling equality in resolution-based procedures and discuss various versions of the paramodulation rule. In Section 1.4 we introduce the main sequent calculus for equality reasoning $\mathbf{LK}^=$ which corresponds to the ground version of semantic tableaux.

1.1. Some useful notation

We assume some knowledge of most fundamental notions of mathematical logic, like those of a term or a formula. They may be found in the standard textbooks on mathematical logic [e.g. Kleene 1952, Smullyan 1968, Lee and Chang 1973, Gallier 1986, Fitting 1990].

- *Formulas* are defined as usual, using atomic formulas, the connectives $\wedge, \vee, \supset, \neg$ and the quantifiers \forall, \exists .
- A *literal* is either an atomic formula or a negation of an atomic formula.
- A *clause* is a finite multiset of literals, denoted L_1, \dots, L_n .
- The *empty clause* is denoted \square .

Atomic formulas will also be called *atoms*. If L is a literal and $C = L_1, \dots, L_n$ is a clause, then L, C will denote the clause L, L_1, \dots, L_n and similar for C, L . Alternatively, we will sometimes consider a clause L_1, \dots, L_n as the formula $L_1 \vee \dots \vee L_n$.

We use the following notation for substitutions and unification:

- *substitutions* will be denoted by $[x_1 \mapsto t_1, \dots, x_n \mapsto t_n]$;
- the *empty substitution* will be denoted ε ;
- the *domain* of a substitution θ , i.e. the set of variables $\{x \mid x\theta \neq x\}$ will be denoted by $\text{dom}(\theta)$;
- the *application* of a substitution $\theta = [x_1 \mapsto t_1, \dots, x_n \mapsto t_n]$ to any expression E (i.e. term, formula, clause, multiset of formulas etc.) is the expression obtained by the simultaneous replacement of free occurrences of x_1, \dots, x_n by t_1, \dots, t_n , respectively, with renaming, if necessary, those bound variables in E that coincide with a variable occurring in any t_i . The result of the application is denoted by $E\theta$.
- The *composition* of substitutions σ and θ , denoted $\sigma\theta$, is the substitution defined by $x(\sigma\theta) = (x\sigma)\theta$, for every variable x .
- A substitution σ is *more general* than a substitution θ , denoted $\sigma \leq \theta$, if and only if there exists a substitution τ such that $\sigma\tau = \theta$.
- A term, literal, or clause is called *ground*, if it contains no variables.
- A formula is called *closed*, if it contains no free occurrences of variables.
- The *universal (respectively, existential) closure* of a formula φ , denoted $\forall\varphi$ (respectively, $\exists\varphi$), is the formula $\forall x_1 \dots \forall x_n \varphi$ (respectively, $\exists x_1 \dots \exists x_n \varphi$), where x_1, \dots, x_n are all free variables of φ in the order of their occurrence in φ .
- A substitution σ is called *grounding* for a set of variables V if for every variable $v \in V$ the term $v\sigma$ is ground.

The notation \doteq stands for “equal by definition”. We denote the equality predicate symbol by \approx . For any expression E (like clause, formula or a multiset of formulas), the set $\text{vars}(E)$ is defined as the set of all (free) variables occurring in E .

- A clause C is called an *instance* of a clause D if there is a substitution σ such that $C\sigma = D$;

- A clause C is called a *variant* of a clause D if C is an instance of D and D is an instance of C .
- An *equation* is any formula of the form $s \approx t$;
- A *disequation* is any formula of the form $\neg(s \approx t)$, denoted $s \not\approx t$.

We do not distinguish the formula $s \approx t$ from $t \approx s$.

We denote

- | | |
|--|--|
| ▷ <i>constants</i> by a, b, c, d, e ; | ▷ <i>literals</i> by L, M, N ; |
| ▷ <i>variables</i> by x, y, z, u, v, w ; | ▷ <i>clauses</i> by C, D ; |
| ▷ <i>function symbols</i> by f, g, h ; | ▷ <i>formulas</i> by φ, χ, ψ ; |
| ▷ <i>predicate symbols</i> by P, Q, R ; | ▷ <i>sets or multisets of formulas</i> by |
| ▷ <i>terms</i> by p, q, r, s, t ; | $\Gamma, \Delta, \Sigma, \Xi$; |
| ▷ <i>atoms</i> by A, B ; | ▷ <i>substitutions</i> by $\theta, \sigma, \delta, \rho$, |

maybe with indices.

We denote by

- $\text{mgu}(s, t)$ any idempotent most general unifier of terms s and t ; and
 - $\text{mgu}(\langle s_1, \dots, s_n \rangle, \langle t_1, \dots, t_n \rangle)$ any idempotent simultaneous most general unifier of s_1 and t_1, \dots, s_n and t_n .
 - We assume that all terms are written in a *fixed finite function signature* \mathcal{F} , unless the opposite is explicitly stated. *Constants* are function symbols of arity 0.
- We denote

- $T_{\mathcal{F}}$ the set of ground terms in the signature \mathcal{F} ;
- $T_{\mathcal{F}}(X)$ the set of terms in the signature \mathcal{F} with variables in a set X .

To avoid expressions with many parentheses, we will denote applications of unary functions to arguments without parentheses, for example fgx instead of $f(g(x))$.

1.2. Equality. The first axiomatization

The *equality predicate* \approx plays a special role in automated reasoning. Mathematical theorems often contain the equality predicate and assume that it satisfies special *equality axioms*. The same is true for many areas of computer science where first-order logic is used.

The necessity of handling equality in automated reasoning has been recognized already in the very early papers in the area [Wang 1960, Kanger 1963, Wos, Robinson, Carson and Shalla 1967, Robinson 1968, Darlington 1968, Robinson and Wos 1969]. Equality in first-order logic can be axiomatized by the following *equality axioms*:

- *reflexivity axiom*: $x \approx x$;
- *symmetry axiom*: $x \approx y \supset y \approx x$;
- *transitivity axiom*: $x \approx y \wedge y \approx z \supset x \approx z$;
- *function substitution axioms*: $x_1 \approx y_1 \wedge \dots \wedge x_n \approx y_n \supset f(x_1, \dots, x_n) \approx f(y_1, \dots, y_n)$, for every function symbol f ;

- *predicate substitution* axioms: $x_1 \approx y_1 \wedge \dots \wedge x_n \approx y_n \wedge P(x_1, \dots, x_n) \supset P(y_1, \dots, y_n)$ for every predicate symbol P .

In principle, for a theorem φ with equality one can add the conjunction χ of these equality axioms as a premise and try to prove $\forall \chi \supset \varphi$ by any existing method for logic without equality. However, this leads to a combinatorial explosion due to the universal applicability of equality axioms.

1.1. EXAMPLE. Suppose that \mathcal{F} contains a binary function symbol f . Then from $a \approx b$ we can derive any equation of the form $f(s_1, f(s_2, \dots, f(s_{n-1}, s_n) \dots)) \approx f(t_1, f(t_2, \dots, f(t_{n-1}, t_n) \dots))$ such that $\{s_1, \dots, s_n, t_1, \dots, t_n\} \subseteq \{a, b\}$. For example, $f(a, f(a, b)) \approx f(a, f(b, a))$ has the following derivation by positive hyperresolution:

$$\frac{a \approx b \quad b \approx a \quad x_1 \not\approx y_1 \vee x_2 \not\approx y_2 \vee f(x_1, x_2) \approx f(y_1, y_2)}{a \approx a \quad f(a, b) \approx f(b, a) \quad x_1 \not\approx y_1 \vee x_2 \not\approx y_2 \vee f(x_1, x_2) \approx f(y_1, y_2)} \quad f(a, f(a, b)) \approx f(a, f(b, a))$$

where $b \approx a$ and $a \approx a$ have the following derivations:

$$\frac{a \approx b \quad x \not\approx y \vee y \approx x}{b \approx a} ;$$

$$\frac{a \approx b \quad b \approx a \quad x \not\approx y \vee y \not\approx z \vee x \approx z}{a \approx a} .$$

Thus, even the early methods of handling equality in automated reasoning tried to avoid the use of the equality axioms. In the rest of this section we will consider some of the main ideas developed in resolution-based theorem proving with equality.

1.3. Paramodulation and its refinements

- We write $\varphi[s]$ to indicate that the formula φ contains an occurrence of the term s such that all occurrences of variables in s are free in φ .

We also denote by $\varphi[t]$ the result of replacing this particular occurrence of s in φ by t . We will use similar notation for occurrences of subterms in terms $r[s]$ and terms in sets or multisets of formulas $\Gamma[s]$.

In first-order logic, the rule or replacement of equal by equal is a derived rule:

$$\frac{\varphi[s] \quad s \approx t}{\varphi[t]} .$$

A suitable formulation of this rule can alternatively be used to replace all equality axioms except for reflexivity. Later, we will give some sequent systems for first-order logic using such a replacement rule. In automated reasoning, a rule of this form has already been used in [Wang 1960].

A modification of this rule for resolution-based reasoning known as *paramodulation* has been introduced by Robinson and Wos [1969]. The paramodulation rule is defined on clauses and uses most general unifiers.

► *Paramodulation* is the following inference rule:

$$\frac{L[s'] \vee C_1 \quad s \approx t \vee C_2}{(L[t] \vee C_1 \vee C_2)\text{mgu}(s, s')} \text{ (par)} \quad (1.1)$$

Robinson and Wos [1969] proved completeness of the system consisting of resolution, factoring and paramodulation only in presence of an additional

► *function reflexivity axiom*: $f(x_1, \dots, x_n) \approx f(x_1, \dots, x_n)$, for every function symbol $f \in \mathcal{F}$.

This axiom seems very similar to the reflexivity axiom $x \approx x$, but there is a fundamental difference important for understanding equality reasoning in general. In order to explain the meaning of the function reflexivity axiom we first show that paramodulation rule leads to *nonliftable derivations*. A standard technique of proving completeness of various methods or strategies in automated deduction is to prove the existence of a ground derivation and then to make *lifting* to the nonground case. Lifting for an inference system \mathcal{R} is a technique for proving completeness that can abstractly be explained in the following way.

► Let an inference system \mathcal{R} have the following property. Suppose that C_1, \dots, C_n be clauses and C'_1, \dots, C'_n be their ground instances. Furthermore, suppose that

$$\frac{C'_1 \quad \dots \quad C'_n}{C'}$$

is an inference of \mathcal{R} , where C' is a ground clause. Then there exists a clause C such that C' is an instance of C and such that C is derivable from C_1, \dots, C_n in \mathcal{R} . Then \mathcal{R} is said to have the *lifting property*.

The logical system consisting of resolution and factoring, as well as many modifications of resolution, has the lifting property. However, the addition of paramodulation leads to nonliftable derivations. Indeed, consider two clauses $P(x, x)$ and $a \approx b$ and their ground instances $P(fa, fa)$ and $a \approx b$. Consider the following ground derivation:

$$\frac{P(fa, fa) \quad a \approx b}{P(fa, fb)} \text{ (par)} \quad (1.2)$$

This derivation is nonliftable.

The system consisting of resolution, factoring, paramodulation and the *function reflexivity axiom* has the lifting property. For example, in this system the above derivation can be lifted as follows, using the axiom $fx \approx fx$:

$$\frac{P(x, x) \quad \frac{fx \approx fx \quad a \approx b}{fa \approx fb} \text{ (par)}}{P(fa, fb)} \text{ (par)}$$

Although function reflexivity leads to a liftable system, it is hard to find any reasonable implementation of function reflexivity, as Example 1.1 demonstrates. As it was shown later, resolution with factoring and paramodulation is complete *without* function reflexivity [Brand 1975, Peterson 1983]. However, the understanding of the nature of function reflexivity is important for understanding some problems arising in automating sequent systems with equality. The effect of the function reflexivity axioms is *the possibility of substituting almost an arbitrary term for a variable*. For example, to lift ground derivation (1.2), we had to substitute fa for x , which can be achieved by using the instance $fx \approx fx$ of the function reflexivity axiom.

Getting rid of function reflexivity drastically reduces search space in methods based on resolution and paramodulation. However, unrestricted applications of paramodulation also lead to combinatorial explosion because of a too general applicability of paramodulation. Further history of paramodulation-based theorem proving had been aiming at restricting the applicability of the paramodulation rule. Among the restrictions, we will briefly consider the following:

1. paramodulation into a variable is prohibited;
2. the use of reduction orderings;
3. the basic strategy of paramodulation;
4. simplification.

Many of these refinements of paramodulation aimed at the development of “a refutation complete set of inference rules for all first-order logic with equality which reduces to the Knuth-Bendix procedure when restricted to equality units” [Peterson 1983]. This refers to the famous completion procedure of Knuth and Bendix [1970] for solving word problems in universal algebras.

The history of the development of these and other paramodulation restrictions can be seen from [Brand 1975, Degtyarev 1979, Degtyarev 1982, Peterson 1983, Degtyarev and Voronkov 1986, Hsiang and Rusinowitch 1986, Zhang and Kapur 1988, Bachmair and Ganzinger 1990, Hsiang and Rusinowitch 1991, Pais and Peterson 1991, Rusinowitch 1991, Bachmair, Ganzinger, Lynch and Snyder 1992, Nieuwenhuis and Rubio 1992b, Bachmair and Ganzinger 1994, Bachmair, Ganzinger, Lynch and Snyder 1995, Nieuwenhuis and Rubio 1995, Lynch 1997, Bachmair and Ganzinger 1998].

We can avoid applying *paramodulation into a variable*:

$$\frac{L[x] \vee C_1 \quad s \approx t \vee C_2}{(L[t] \vee C_1 \vee C_2)[x \mapsto s]} \text{ (par)}$$

Note that such an application of paramodulation is *always possible* when we have a nonground clause $L[x] \vee C_1$, because x is unifiable with s . It is known that the system consisting of paramodulation, resolution and factoring is complete when paramodulation into a variable is prohibited (i.e. the term s' in paramodulation rule (1.1) is not a variable) [Brand 1975, Peterson 1983]¹

¹Strictly speaking, neither Brand nor Peterson could entirely prohibit paramodulations into variables. This problem was completely solved later by Bachmair et al. [1992] and Nieuwenhuis and Rubio [1992b].

Often, paramodulation allows one to repeatedly apply the same equation obtaining larger and larger terms.

1.2. EXAMPLE. The equation $x \approx fx$ can repeatedly be applied as in the derivation below:

$$\frac{\frac{Pa \quad x \approx fx}{P(fa)} (par)}{P(ffa)} (par) \\ \vdots \\ P(f \dots fa)$$

obtaining longer and longer literals $P(f^n a)$.

Such *increasing applications* of paramodulations can be avoided by the introduction of ordering restrictions based on *reduction orderings*.

- A *reduction ordering* on $T_{\mathcal{F}}(X)$ is any ordering \succ on $T_{\mathcal{F}}(X)$ such that
 1. \succ is well-founded;
 2. if $s \succ t$ then $r[s\sigma] \succ r[t\sigma]$, for all terms s, t, r and substitutions σ .
- A reduction ordering is called a *simplification ordering* if for any term $t[s]$ such that s is a proper subterm of t we have $t[s] \succ s$.

The results involving reduction orderings in this chapter are true for any reduction ordering total on $T_{\mathcal{F}}$.

Reduction orderings are used for two main purposes: restricting paramodulation and simplification. Here and below we write

- $s \succeq t$ to denote that $s \succ t$ or $s = t$.

The simplest ordering restriction on paramodulation gives us

- *ordered paramodulation* that can be formulated as follows:

$$\frac{L[s'] \vee C_1 \quad s \approx t \vee C_2}{(L[t] \vee C_1 \vee C_2)\sigma}$$

such that

1. $\sigma = \text{mgu}(s, s')$;
2. s' is not a variable;
3. $t\sigma \not\succeq s\sigma$.

Note that when $s\sigma$ and $t\sigma$ are ground terms and \succ is linear on ground terms, the last condition is equivalent to $s\sigma \succ t\sigma$. Ordered paramodulation has been originally considered in [Peterson 1983] and [Hsiang and Rusinowitch 1986].

If we use ordered paramodulation, no derivation from $P(a)$ and $x \approx fx$ is possible (compare with Example 1.2).

A stronger restriction is the so-called *maximal paramodulation* (in the terminology of [Pais and Peterson 1991]) that takes into attention not only ordering on terms but also ordering on literals of clauses.

► *Maximal paramodulation* is the following inference rule:

$$\frac{L[s'] \vee C_1 \quad s \approx t \vee C_2}{(L[t] \vee C_1 \vee C_2)}$$

such that

1. $\sigma = \text{mgu}(s, s')$;
2. term s' is not a variable;
3. $t\sigma \not\approx s\sigma$;
4. $L[s']\sigma$ is maximal w.r.t. \succ in $(L[s'] \vee C_1)\sigma$;
5. $(s \approx t)\sigma$ is maximal w.r.t. \succ in $(s \approx t \vee C_2)\sigma$.

The next refinement of paramodulation is known as *superposition*. The notion of superposition comes from Knuth and Bendix [1970], where superposition has been defined for positive unit clauses (equations). The above definition of maximal paramodulation can be strengthened to the definition of superposition in the following way. For simplicity, we assume that all atoms are equations. Then, the literal $L[s']$ in the paramodulation rule has either the form $p[s'] \approx q$ or $p[s'] \not\approx q$. We only consider the former case.

► *Superposition* is the following inference rule:

$$\frac{p[s'] \approx q \vee C_1 \quad s \approx t \vee C_2}{(p[t] \approx q \vee C_1 \vee C_2)\sigma}$$

such that

1. $\sigma = \text{mgu}(s, s')$;
2. s' is not a variable;
3. $t\sigma \not\approx s\sigma$;
4. $(p[s'] \approx q)\sigma$ is maximal w.r.t. \succ in $(p[s'] \approx q \vee C_1)\sigma$;
5. $(s \approx t)\sigma$ is maximal w.r.t. \succ in $(s \approx t \vee C_2)\sigma$;
6. $q\sigma \not\approx p[s']\sigma$.

A calculus with such a rule was proposed by Zhang and Kapur [1988]. If we drop condition (5), we come to *extended superposition* [Rusinowitch 1991]. There are further refinements of superposition, for example *strict superposition* of [Bachmair and Ganzinger 1990, Bachmair and Ganzinger 1991, Bachmair and Ganzinger 1998].

Recent results in this area are related to the so-called *basic strategy*. The idea of the basic strategy is to forbid paramodulation in terms introduced by substitutions applied during the previous inference steps. The basic strategy (without ordering restrictions) has been explicitly introduced for the first time in [Degtyarev 1979, Degtyarev 1982] by the name of *monotone paramodulation*. It has also been described in [Degtyarev and Voronkov 1986] in terms of the so-called *conditional clauses* which are now known by the name of *equality constraint clauses* [Nieuwenhuis and Rubio 1992a]. Such clauses have been defined in [Degtyarev and Voronkov 1986] as pairs $C \cdot \mathcal{C}$ where C is a clause and \mathcal{C} is a set of equations $\{s_1 \approx t_1, \dots, s_n \approx t_n\}$. The equations $s_i \approx t_i$ in \mathcal{C} are considered as constraints to

be solved by a *simultaneous most general unifier* of (s_i, t_i) , i.e. a substitution σ such that $s_i\sigma = t_i\sigma$ for all $i \in \{1, \dots, n\}$. Thus, the substitution σ introduced by the previous inference steps has been divided from the “skeleton” C of the clause $C\sigma$. Following Degtyarev [1982], Degtyarev and Voronkov [1986] described two forms of *monotone paramodulation*:

$$\frac{L[s'] \vee C_1 \cdot C_1 \quad s \approx t \vee C_2 \cdot C_2}{L[t] \vee C_1 \vee C_2 \cdot C_1 \cup C_2 \cup \{s' \approx s\}}$$

and

$$\frac{L[s'] \vee C_1 \cdot C_1 \quad s \approx t \vee C_2 \cdot C_2}{L[z] \vee C_1 \vee C_2 \cdot C_1 \cup C_2 \cup \{s' \approx s, z \approx t\}}$$

where in both rules s' is not a variable and in the second rule z is a new variable².

The basic strategy with ordering restrictions has later been independently proposed in [Nieuwenhuis and Rubio 1992a, Nieuwenhuis and Rubio 1992b] and [Bachmair et al. 1992] as an extension of a *basic narrowing* technique proposed first by Hullot [1980] in the Knuth-Bendix framework to refutational theorem proving with arbitrary clauses. In these papers the basic strategy has been applied to superposition, i.e. combined with ordering restrictions. The ordering restrictions can be elegantly added by extending equational constraints to *ordering constraints*. A variant of basic superposition with *ordering constraint inheritance* proposed in [Nieuwenhuis and Rubio 1992b, Nieuwenhuis and Rubio 1995] is defined and used in Section 6.2. Bachmair et al. [1995] described further refinements of the basic strategy which included *term selection functions* and *redex ordering*.

Resolution-based theorem proving is usually based on the *saturation procedures*. Such procedures start with a set of clauses S , adding to this set new clauses obtained by application of inference rules to clauses in S . The search space for such procedures grows rapidly. Saturation procedures become much more efficient when they are augmented with *redundancy criteria* allowing us to remove redundant clauses from the search space. Examples of redundancy deletion rules are *subsumption* and *simplification*. Subsumption was introduced by Robinson [1965] for the general resolution and simplification has been introduced for equality reasoning by Knuth and Bendix [1970]. In order to formalize simplification, we introduce inference rules working on multisets of clauses. Suppose S is a multiset of clauses. Then

► *simplification* is the following inference rule:

$$\frac{S \cup \{L[s'] \vee C, s \approx t\}}{S \cup \{L[t\sigma] \vee C, s \approx t\}} \quad (1.3)$$

such that $s\sigma = s'$, $s\sigma \succ t\sigma$ and $L[s\sigma] \succ (s\sigma \approx t\sigma)$ [Peterson 1983].

²In the second form, after paramodulation into s' , subsequent paramodulations are only possible into positions “orthogonal” to s' or into positions of superterms of s' in $L[s']$. Hence the name “monotone paramodulation”.

The meaning of this inference rule is that s' can be *replaced* by $t\sigma$, thus discarding the clause $L[s'] \vee C$. Simplification has been recognized a very strong strategy for equational reasoning. However, until very recently, simplification has not been introduced in sequent-based methods. There are other formulations of simplification, where the condition $L[s\sigma] \succ (\sigma \approx t\sigma)$ is replaced by other conditions (e.g. [Rusinowitch 1991] and [Bachmair and Ganzinger 1994]). As far as we know, these conditions are needed in order to apply the corresponding completeness proofs. It is not known whether these conditions are necessary.

Discussion of various aspects of redundancy criteria can be found in [Wos et al. 1967, Knuth and Bendix 1970, Slagle 1974, Lankford 1975, Loveland 1978, Peterson 1983, Wos, Overbeek and Lusk 1991, Rusinowitch 1991, Lusk 1992, Voronkov 1992, Bachmair and Ganzinger 1994, Bachmair et al. 1995, Nieuwenhuis and Rubio 1995, Tammet 1996, Mints, Orevkov and Tammet 1996, Lynch 1997, Bachmair and Ganzinger 1998].

In all examples of this chapter we will use a special kind of reduction ordering, called the *lexicographic path ordering*.

- A *precedence relation* on \mathcal{F} is any total ordering on \mathcal{F} .
- Let $\succ_{\mathcal{F}}$ be a precedence relation on \mathcal{F} . The *lexicographic path ordering* \succ induced by $\succ_{\mathcal{F}}$ is the ordering on terms defined recursively as follows:

$t \succ x$ if $x \neq t$ and x occurs in t ;

It is not the case that $x \succ t$;

Let $s = f(s_1, \dots, s_m)$ and $t = g(t_1, \dots, t_n)$. Then $s \succ t$ if one of the following is true:

$s_i \succeq t$, for some i with $1 \leq i \leq m$;

$f \succ_{\mathcal{F}} g$ and $s \succ t_j$, for all j with $1 \leq j \leq n$;

$f = g$, $(s_1, \dots, s_m) \succ^{lex} (t_1, \dots, t_n)$ and $s \succ t_j$, for all j with $1 \leq j \leq n$, where $(s_1, \dots, s_m) \succ^{lex} (t_1, \dots, t_m)$ if there is $j \leq m$ such that $s_j \succ t_j$ and for all $i < j$ we have $s_i = t_i$.

The reader can check that lexicographic path ordering is a simplification ordering, and that it is total on $\mathcal{T}_{\mathcal{F}}$ when $\succ_{\mathcal{F}}$ is total on \mathcal{F} .

1.4. Equality in sequent systems

In this section we introduce the sequent calculus $\mathbf{LK}^=$ for classical first-order logic with equality. We also explain the relation between ground version of semantic tableaux and derivations in $\mathbf{LK}^=$.

- A *sequent* is any expression of the form $\Gamma \rightarrow \Delta$, where Γ, Δ are finite multisets of formulas.
- Γ (respectively Δ) is called the *antecedent* (respectively, the *succedent*) of the sequent $\Gamma \rightarrow \Delta$.
- A sequent $\Gamma \rightarrow \Delta$ is *closed* if all formulas in Γ, Δ are closed.

We will denote sequents by S , maybe with indices.

- The *sequent calculus* $\mathbf{LK}^=$ is shown in Figure 1.

$$\begin{array}{c}
\overline{\Gamma, A \rightarrow \Delta, A} \quad (Ax) \\
\\
\overline{\Gamma \rightarrow \Delta, t \approx t} \quad (refl) \qquad \frac{\Gamma[x \mapsto t], s \approx t \rightarrow \Delta[x \mapsto t]}{\Gamma[x \mapsto s], s \approx t \rightarrow \Delta[x \mapsto s]} \quad (\approx) \\
\\
\frac{\Gamma, \varphi, \psi \rightarrow \Delta}{\Gamma, \varphi \wedge \psi \rightarrow \Delta} \quad (\wedge \rightarrow) \qquad \frac{\Gamma \rightarrow \Delta, \varphi \quad \Gamma \rightarrow \Delta, \psi}{\Gamma \rightarrow \Delta, \varphi \wedge \psi} \quad (\rightarrow \wedge) \\
\\
\frac{\Gamma, \varphi \rightarrow \Delta \quad \Gamma, \psi \rightarrow \Delta}{\Gamma, \varphi \vee \psi \rightarrow \Delta} \quad (\vee \rightarrow) \qquad \frac{\Gamma \rightarrow \Delta, \varphi, \psi}{\Gamma \rightarrow \Delta, \varphi \vee \psi} \quad (\rightarrow \vee) \\
\\
\frac{\Gamma, \psi \rightarrow \Delta \quad \Gamma \rightarrow \Delta, \varphi}{\Gamma, \varphi \supset \psi \rightarrow \Delta} \quad (\supset \rightarrow) \qquad \frac{\varphi, \Gamma \rightarrow \Delta, \psi}{\Gamma \rightarrow \Delta, \varphi \supset \psi} \quad (\rightarrow \supset) \\
\\
\frac{\Gamma \rightarrow \Delta, \varphi}{\Gamma, \neg \varphi \rightarrow \Delta} \quad (\neg \rightarrow) \qquad \frac{\varphi, \Gamma \rightarrow \Delta}{\Gamma \rightarrow \Delta, \neg \varphi} \quad (\rightarrow \neg) \\
\\
\frac{\Gamma, \forall x \varphi, \varphi[x \mapsto t] \rightarrow \Delta}{\Gamma, \forall x \varphi \rightarrow \Delta} \quad (\forall \rightarrow) \qquad \frac{\Gamma \rightarrow \Delta, \varphi[x \mapsto y]}{\Gamma \rightarrow \Delta, \forall x \varphi} \quad (\rightarrow \forall) \\
\\
\frac{\Gamma, \varphi[x \mapsto y] \rightarrow \Delta}{\Gamma, \exists x \varphi \rightarrow \Delta} \quad (\exists \rightarrow) \qquad \frac{\Gamma \rightarrow \Delta, \exists x \varphi, \varphi[x \mapsto t]}{\Gamma \rightarrow \Delta, \exists x \varphi} \quad (\rightarrow \exists)
\end{array}$$

In the rule (Ax) , A is an atomic formula. In the rules $(\rightarrow \forall)$ and $(\exists \rightarrow)$ the variable y has no free occurrences in the conclusions of the rules. The variable y is called the *eigenvariable* of these rules. This condition on the rules $(\rightarrow \forall)$ and $(\exists \rightarrow)$ is called the *eigenvariable condition*.

Figure 1: Calculus \mathbf{LK}^-

This calculus is similar to the calculi introduced in [Kanger 1957, Kanger 1963].

The intuitive semantics of a sequent $\varphi_1, \dots, \varphi_n \rightarrow \psi_1, \dots, \psi_m$ is $\bigwedge_{i \in \{1, \dots, n\}} \varphi_i \supset \bigvee_{j \in \{1, \dots, m\}} \psi_j$. Thus, this sequent is inconsistent if and only if all φ_i are true and all ψ_j are false. For this reason, instead of sequents one can dually consider multisets of *signed formulas*, where *signs* T and F correspond to “true” and “false”. For example, the above sequent can alternatively be represented as the multiset of signed formulas $T \varphi_1, \dots, T \varphi_n, F \psi_1, \dots, F \psi_m$. Conventional formalizations of the tableau method deal with signed formulas. For convenience, we will sometimes alternatively consider sequents as multisets of signed formulas. Multisets of signed formulas will also be denoted by Γ, Δ , maybe with indices.

In view of the relation between sequents and multisets of signed formulas, the calculus \mathbf{LK}^- can be reformulated in terms of signed formulas. Inference rules working on signed formulas are traditionally classified into α -, β -, γ - and δ -rules.

The correspondence is given in the following table:

$$\begin{array}{ccc}
\frac{}{\Gamma, T A, F A} (Ax) & \frac{}{\Gamma, F t \approx t} (refl) & \frac{\Gamma[x \mapsto t], T s \approx t}{\Gamma[x \mapsto s], T s \approx t} (\approx) \\
\\
\frac{\Gamma, T \varphi, T \psi}{\Gamma, T \varphi \wedge \psi} (\alpha) & & \frac{\Gamma, F \varphi \quad \Gamma, F \psi}{\Gamma, F \varphi \wedge \psi} (\beta) \\
\\
\frac{\Gamma, T \varphi \quad \Gamma, T \psi}{\Gamma, T \varphi \vee \psi} (\beta) & & \frac{\Gamma, F \varphi, F \psi}{\Gamma, F \varphi \vee \psi} (\alpha) \\
\\
\frac{\Gamma, T \psi \quad \Gamma, F \varphi}{\Gamma, T \varphi \supset \psi} (\beta) & & \frac{\Gamma, T \varphi, F \psi}{\Gamma, F \varphi \supset \psi} (\alpha) \\
\\
\frac{\Gamma, F \varphi}{\Gamma, T \neg \varphi} (\neg) & & \frac{\Gamma, T \varphi}{\Gamma, F \neg \varphi} (\neg) \\
\\
\frac{\Gamma, T \forall x \varphi, T \varphi[x \mapsto t]}{\Gamma, T \forall x \varphi} (\gamma) & & \frac{\Gamma, F \varphi[x \mapsto y]}{\Gamma, F \forall x \varphi} (\delta) \\
\\
\frac{\Gamma, T \varphi[x \mapsto y]}{\Gamma, T \exists x \varphi} (\delta) & & \frac{\Gamma, F \exists x \varphi, F \varphi[x \mapsto t]}{\Gamma, F \exists x \varphi} (\gamma)
\end{array}$$

In the rule (Ax) , A is an atomic formula. Both rules (δ) satisfy the eigenvariable condition.

Figure 2: Calculus $\mathbf{LK}^=$ for signed formulas

Signed formulas	Sequents
α	$(\wedge \rightarrow), (\rightarrow \vee), (\rightarrow \supset), (\rightarrow \neg), (\neg \rightarrow)$
β	$(\rightarrow \wedge), (\vee \rightarrow), (\supset \rightarrow)$
γ	$(\forall \rightarrow), (\rightarrow \exists)$
δ	$(\rightarrow \forall), (\exists \rightarrow)$

The calculus $\mathbf{LK}^=$ using signed formulas is shown in Figure 2.

The ordinary $\mathbf{LK}^=$ derives sequents, $\mathbf{LK}^=$ for signed formulas derives multisets of formulas. Derivations of $\mathbf{LK}^=$ for signed formulas are sometimes represented in the form of so-called *tableaux* [see Fitting 1990]. We will give examples below.

- We write $\Gamma \vdash \varphi$ to denote that the sequent $\Gamma \rightarrow \varphi$ is derivable in $\mathbf{LK}^=$.
- We speak about *derivations* or *derivability* of a formula φ meaning derivations or derivability of the sequent $\rightarrow \varphi$.
- A derivation Π in $\mathbf{LK}^=$ is called *regular* if it satisfies the following properties:
 1. equality rules are applied before all other rules, i.e. above applications of (\approx) in Π there can only be applications of (\approx) , (Ax) and $(refl)$;

2. in the rules (\approx), the occurrences of t replaced by s are in atomic formulas only.

We give an example which will be used several times in this chapter.

In order to simplify derivations, we will display them simplified in two ways:

1. Sometimes we will use a modified form of the rules ($\rightarrow \exists$) and (γ) without repeating their principal formula $\exists x\varphi$ in the premises, i.e., use the rules

$$\frac{\Gamma \rightarrow \Delta, \varphi[x \mapsto t]}{\Gamma \rightarrow \Delta, \exists x\varphi} (\rightarrow \exists) \quad \text{and} \quad \frac{\Gamma, T \forall x\varphi, T \varphi[x \mapsto t]}{\Gamma, T \forall x\varphi} (\gamma).$$

2. We will group sequences of similar inferences into one inference, for example

$$\frac{\rightarrow \varphi_1 \quad \rightarrow \varphi_2 \quad \rightarrow \varphi_3}{\rightarrow \varphi_1 \wedge \varphi_2 \wedge \varphi_3} (\rightarrow \wedge)$$

can be used to denote the sequence of two ($\rightarrow \wedge$) rule.

1.3. EXAMPLE (*Main example, sequent calculus*). Assume that we want to prove the formula

$$\exists xyuv((a \approx b \supset g(x, u, v) \approx g(y, fc, fd)) \wedge (c \approx d \supset g(u, x, y) \approx g(v, fa, fb))).$$

We show its regular derivation in $\mathbf{LK}^=$ in the following three forms:

1. In $\mathbf{LK}^=$ using sequents in Figure 3;
2. In $\mathbf{LK}^=$ using multisets of signed formulas (without duplicating) in Figure 4;
3. In a ground version of a tableau system in Figure 5.

Tableaux are trees whose nodes are signed formulas. A tableau can be thought of as a multiset of *branches*, every branch is understood as a multiset of formulas occurring in this branch. A tableau \mathcal{T} represents a $\mathbf{LK}^=$ -derivation D such that the leaves of D are the branches of the tableau. We can note the following inessential difference between tableaux and derivations in $\mathbf{LK}^=$. When we counterapply an α -rule in $\mathbf{LK}^=$, for example

$$\frac{T a \approx b, F g(fa, fc, fd) \approx g(fb, fc, fd)}{F a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd)} (\alpha),$$

we *remove* the principal formula (in this case $F a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd)$) from the premise, while it remains on the tableau branch.

As can be seen from these examples, sequent derivations, corresponding derivations on the multisets of signed formulas and the ground version of semantic tableaux are different ways of representing derivations in $\mathbf{LK}^=$. For the rest of this chapter, we will mostly use $\mathbf{LK}^=$ in the form of signed formulas.

Many techniques considered in this chapter are implicitly based on the fact that we can restrict ourselves by regular derivations:

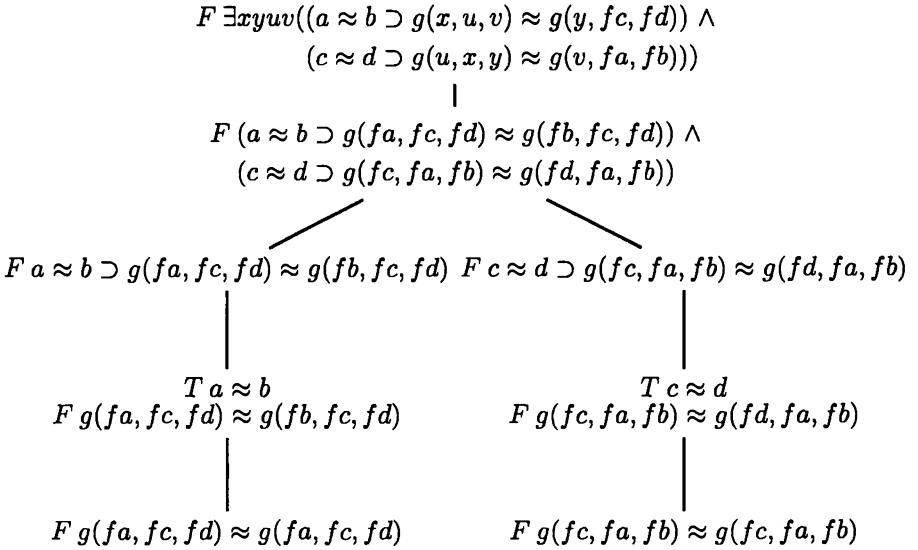
1.4. THEOREM (existence of regular derivations in $\mathbf{LK}^=$). *Any sequent derivable in $\mathbf{LK}^=$ has a regular derivation.*

$$\begin{array}{c}
\frac{a \approx b \rightarrow g(fa, fc, fd) \approx g(fa, fc, fd)}{a \approx b \rightarrow g(fa, fc, fd) \approx g(fb, fc, fd)} \quad (refl) \\
\frac{a \approx b \rightarrow g(fa, fc, fd) \approx g(fb, fc, fd)}{\rightarrow a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd)} \quad (\rightarrow \supset) \\
\frac{\rightarrow a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd) \wedge (c \approx d \supset g(fc, fa, fb) \approx g(fd, fa, fb))}{\rightarrow (a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd)) \wedge (c \approx d \supset g(fc, fa, fb) \approx g(fd, fa, fb))} \quad (\rightarrow \wedge) \\
\frac{\rightarrow (a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd)) \wedge (c \approx d \supset g(fc, fa, fb) \approx g(fd, fa, fb))}{\rightarrow \exists xyuv((a \approx b \supset g(x, u, v) \approx g(y, fc, fd)) \wedge (c \approx d \supset g(u, x, y) \approx g(v, fa, fb)))} \quad (\rightarrow \exists)
\end{array}$$

Figure 3: An $LK^=$ -derivation using sequents

$$\begin{array}{c}
\frac{T a \approx b, F g(fa, fc, fd) \approx g(fa, fc, fd)}{T a \approx b, F g(fa, fc, fd) \approx g(fb, fc, fd)} \quad (refl) \\
\frac{T a \approx b, F g(fa, fc, fd) \approx g(fb, fc, fd)}{F a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd)} \quad (\alpha) \\
\frac{F a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd) \wedge (c \approx d \supset g(fc, fa, fb) \approx g(fd, fa, fb))}{F \exists xyuv((a \approx b \supset g(x, u, v) \approx g(y, fc, fd)) \wedge (c \approx d \supset g(u, x, y) \approx g(v, fa, fb)))} \quad (\beta) \\
\frac{F \exists xyuv((a \approx b \supset g(x, u, v) \approx g(y, fc, fd)) \wedge (c \approx d \supset g(u, x, y) \approx g(v, fa, fb)))}{F \exists xyuv((a \approx b \supset g(x, u, v) \approx g(y, fc, fd)) \wedge (c \approx d \supset g(u, x, y) \approx g(v, fa, fb)))} \quad (\gamma)
\end{array}$$

Figure 4: An $LK^=$ -derivation using signed formulas

Figure 5: An $\mathbf{LK}^=$ -derivation using tableaux

For simplicity, in this chapter we do not consider derivations in the method of matings as described in [Andrews 1976, Andrews 1981, Andrews 1986] or the connection method as described in [Bibel and Schreiber 1975] and [Bibel 1981, Bibel 1993]. For those, who is used to connections or matings, we will give an example derivation of the same formula in Section 3, after having considered the free variable version of semantic tableaux.

- A formula is said to be in *negation normal form* if it is constructed from literals using the connectives \wedge, \vee and the quantifiers \forall, \exists .
- A formula is said to be in *Skolem negation normal form* if it is constructed from literals using the connectives \wedge, \vee and the quantifier \exists .

There is a provability-preserving translation of formulas without equivalences into formulas in skolem negation normal form consisting of the standard skolemization and a translation into negation normal form used, e.g. by Andrews [1981].

Sequent systems based on signed formulas avoid some redundancies by joining similar inference rules under one name, for example, α -rule. For formulas in negation normal form, there are more elegant sequent systems. One of such systems has already been described in [Schütte 1960]. First, for formulas in negation normal form signs are not needed any more. Instead of using signs, we can consider all formulas in a sequent as having the sign F by considering axioms of the form $\Gamma, A, \neg A$ instead of $\Gamma, T A, F A$. Second, there is only one inference rule for each connective: all α -rules become a rule for disjunction, all β -rules become a rule for conjunction,

$$\begin{array}{ccc}
\frac{}{\Gamma, A, \neg A} (Ax) & \frac{}{\Gamma, t \approx t} (refl) & \frac{\Gamma[x \mapsto t], s \not\approx t}{\Gamma[x \mapsto s], s \not\approx t} (\approx) \\
\\
\frac{\Gamma, \varphi, \psi}{\Gamma, \varphi \vee \psi} (\vee) & & \frac{\Gamma, \varphi \quad \Gamma, \psi}{\Gamma, \varphi \wedge \psi} (\wedge) \\
\\
\frac{\Gamma, \varphi[x \mapsto t], \exists x \varphi}{\Gamma, \exists x \varphi} (\exists) & & \frac{\Gamma, \varphi[x \mapsto y]}{\Gamma, \forall x \varphi} (\forall)
\end{array}$$

In the rule (Ax) , A is an atomic formula. The rule (\forall) satisfies the eigenvariable condition.

Figure 6: Calculus $\mathbf{LK}^=$ for formulas in negation normal form

γ - and δ -rules become quantifier rules (see Figure 6). The Schütte system did not even have the (\vee) -rule, because he considered disjunctions of formulas instead of sequents.

For formulas in skolem negation normal forms, we do not need the (\vee) -rule. Sequent systems for formulas in skolem negation normal forms can further be refined. For example in [Voronkov 1990b] a sequent system is defined whose only rule is the (\wedge) -rule.

In this chapter, when it concerns classical logic, we only work with skolemized formula to avoid the use of δ -rules (i.e. the rules $(\rightarrow \vee)$ and $(\exists \rightarrow)$) of $\mathbf{LK}^=$. Various forms of δ -rules have been considered in [Beckert, Hähnle and Schmitt 1993, Hähnle and Schmitt 1994, Baaz and Leitsch 1994, Baaz and Fermüller 1995]. We cannot, however, avoid using δ -rules in sequent calculi for intuitionistic logic (see Section 9).

1.5. Overview of calculi introduced in this chapter

In this chapter, we introduce many different calculi. Following a suggestion of Nikolai Björner, we provide an overview of all the calculi of this chapter in Tables 1 and 2.

2. Translation of logic with equality into logic without equality

Instead of inventing methods of equality handling in sequent calculi, one can use “efficient” transformations of logic with equality into logic without equality. By “efficient” we mean that the transformations avoid adding equality axioms. There are two such methods, the first is proposed by Brand [1975] and known as the modification method, or *Brand’s transformation* and the second by Moser and Steinbach [1997] and independently in a stronger form by Bachmair, Ganzinger

calculus	page	features
$LK^=$	623	The standard sequent calculus for classical logic with equality with invertible rules.
$LK^=$	624	The same calculus, but using signed formulas instead of sequents with antecedents and succedents.
$LK^=$	628	The same calculus, but for formulas in negation normal form. A sequent in this calculus is a multiset of formulas.
TK	638	The standard free-variable tableau calculus for classical logic. Derived tableaux, that is multisets of sequents. Uses most general unifiers for instantiating variables.
IK_ξ	643	The free-variable calculus for the inverse method (without equality). The free-variable version of $LK^=$ intended for the forward proof-search.
BSE	655	The calculus of basic superposition and equality solution for solving rigid equations. Based on a “rigid” versions of superposition. Sound but not complete for solving rigid equations.
$TK^=$	661	Fitting’s calculus that extends TK to handle equality. In addition to the rules of TK , contains the tableau paramodulation rule, the free variable tableau reflexivity rule and the tableau function reflexivity rule.
$TBSE$	664	The tableau calculus. A mixture of TK and BSE : the rules of TK are also the rules of $TBSE$, the rules of BSE are adapted to tableaux. Derives constraint tableaux, that is tableaux with inequality and ordering constraints.
RRP	665	The rigid version of the resolution calculus. Uses rigid versions of the standard rules: resolution, factoring, reflexivity and paramodulation.
BS_ξ	672	The basic superposition calculus specialized to the (subformulas of) the goal formula ξ . Derives so-called solution clauses. The solution clauses are enough to close tableau branches without further applications of equality rules.
T_ξ	675	The tableau calculus for the equality elimination method. Consists of the standard rules TK specialized to the goal formula ξ plus the branch closure rule that closes tableau branches by the instances of solution clauses using subset unification. Uses the naming technique: sequents consist of atomic names encoding the subformulas of the goal formula.
I_ξ	677	The sequent calculus for the inverse method with equality. Uses the naming technique and solution clauses as well. Unlike T_ξ , does not treat variables rigidly.

Table 1: Overview of calculi used in this chapter

calculus	page	features
$\mathbf{LJ}^=$	680	The standard sequent calculus for intuitionistic logic with equality. Not a free-variable calculus.
$\mathbf{LJ}_c^=$	687	The free-variable calculus for intuitionistic logic with equality “modulo simultaneous rigid E -unification”. Uses sequents with constraints. Simultaneous rigid E -unification problems can be parts of the constraints.
$\mathbf{S4}^=$	690	The sequent calculus for the modal logic $S4$ with equality.

Table 2: Overview of calculi used in this chapter (continued)

and Voronkov [1998], under the name of *constraint equality elimination*. Since these transformations are used to implement equality in several tableau-based theorem provers [Schumann 1994, Bibel, Brüning, Egly, Korn and Rath 1995], we briefly consider them in this section.

Both transformations are applied to a set of clauses. To apply them to a set of formulas, one should first transfer them to the clause form. Both transformations apply to the set of clauses a sequence of transformation steps, each step eliminating the need in some equality axioms. Throughout this section we assume that we deal with a set of clauses whose only predicate symbol is equality.

2.1. Modification method

The transformation used by the *modification method* comprises three parts: flattening, symmetry elimination and transitivity elimination.

Flattening. This transformation eliminates the need in function and predicate substitution axioms.

- A clause is *flat* if all occurrences of non-variable terms in it are arguments to the equality predicate.
- A *flat form of a clause* C is any clause that is flat and is equivalent to C .

For example, the clause $g(a, b) \approx b$ is not flat. Two possible flat forms of this clause are $b \not\approx x \vee a \not\approx z \vee g(z, x) \approx x$ and $b \not\approx x \vee a \not\approx z \vee b \not\approx y \vee g(z, x) \approx y$.

Every clause can be translated to an equivalent flat clause by the flattening transformation:

$$C[f(t)] \Rightarrow t \not\approx y \vee C[f(y)],$$

where y is a new variable. The essence of flattening is expressed by the following theorem of Brand [1975].

2.1. THEOREM. *A set of flat clauses is satisfiable if and only if it has a model in which \approx is interpreted as an equivalence relation.*

In this section, when we speak about unsatisfiability of a set of clauses, we will treat the equality predicate in two different ways. One possibility is to define satisfiability as the existence of a model satisfying all clauses *and the equality axioms*. In this case we will speak about *equational satisfiability*. Another possibility is to treat the equality predicate as satisfying either some equality axioms or none at all. Flattening eliminates the need in function and predicate substitution axioms, thus leaving us only with reflexivity, symmetry and transitivity. Thus, the above theorem says that a set of clauses is equationally unsatisfiable if and only if it has no model in which \approx is an equivalence relation.

Thus, instead of adding all equality axioms to the set of flattened clauses we have to only add the axioms expressing that \approx is an equivalence relation:

$$\begin{aligned} x &\approx x, \\ x \not\approx y \vee y &\approx x, \\ x \not\approx y \vee y \not\approx z \vee x &\approx z. \end{aligned}$$

The next two steps of the transformation allow us to get rid of symmetry and transitivity.

Symmetry elimination.

- We call a *symmetric version* of C any clause obtained from C by replacing one or more positive equations $s \approx t$ by $t \approx s$.
- The *symmetry elimination* replaces every clause by all symmetric versions of this clause.

For example, one possible symmetric version of the clause $a \approx b \vee b \approx c \vee x \not\approx f(x)$ is $a \approx b \vee c \approx b \vee x \not\approx f(x)$. The clause $a \approx b \vee b \approx c \vee f(x) \not\approx x$ is *not* a symmetric version of this clause, since only the sides of positive equations can be swapped. Evidently, symmetry elimination produces 2^n clauses from a clause with n positive equations (unless the clause contains a literal $t \approx t$, in which case it is redundant).

One can prove that after symmetry elimination only transitivity and reflexivity axioms should be taken care of:

2.2. THEOREM. *Let N be a set of flat clauses and N' be obtained from N by symmetry elimination. Then N is equationally unsatisfiable if and only if N' has no model in which \approx is interpreted as a reflexive and transitive relation.*

Transitivity elimination. Transitivity elimination replaces every positive equation $s \approx t$ by the disjunction $t \not\approx y \vee s \approx y$ with a new auxiliary variable y .

For example, transitivity elimination replaces the clause

$$f(x) \approx a \vee b \approx y \vee x \not\approx y$$

by the clause

$$a \not\approx z \vee f(x) \approx z \vee y \not\approx u \vee b \approx u \vee x \not\approx y.$$

After transitivity elimination the only axiom that remains is reflexivity:

2.3. THEOREM. *Let N be a set of flat clauses and N' be obtained from N by symmetry and transitivity elimination. Then N is equationally unsatisfiable if and only if $N' \cup \{x \approx x\}$ is unsatisfiable.*

Consider an example. Suppose that the modification method is applied to the following set of two clauses:

$$b \approx c \quad (2.1)$$

$$f(c, x) \not\approx x \quad (2.2)$$

Clause (2.1) is flat. Symmetry and transitivity elimination applied to this clause yield clauses (2.3) and (2.4) below. Clause (2.2) is not flat. Flattening yields clause (2.5) below. Symmetry and transitivity elimination do not change the clause. Thus, the modification method gives the following four clauses.

$$b \not\approx y \vee c \approx y \quad (2.3)$$

$$c \not\approx y \vee b \approx y \quad (2.4)$$

$$c \not\approx y \vee f(y, x) \not\approx x \quad (2.5)$$

$$x \approx x \quad (2.6)$$

Brand's transformation and basic paramodulation. It was noted by several researchers that there exist close connections between the modification method and basic paramodulation. Basic paramodulation inferences on the set of original clauses can be simulated by resolution on the set of transformed clauses. Consider, for example, the following paramodulation inference from clauses (2.1) and (2.2):

$$\frac{b \approx c \quad f(c, x) \not\approx x}{f(b, x) \not\approx x}.$$

If we transform, using the modification method, the conclusion of this inference, we obtain the clause $b \not\approx y \vee f(y, x) \not\approx x$. The same clause can be obtained by the resolution inference from modified clauses (2.3) and (2.5)

$$\frac{b \not\approx y \vee c \approx y \quad c \not\approx y \vee f(y, x) \not\approx x}{b \not\approx y \vee f(y, x) \not\approx x}.$$

This technique of simulating derivations in the paramodulation-based refutation system by resolution-based derivations on the transformed set of clauses can be used to prove completeness of the modification method. Brand used pure model-theoretic technique to prove completeness.

One can also note that some resolution inferences on the modified set of clauses correspond to redundant inferences on the original set of clauses. For example, paramodulation into a variable is known to be redundant. The picture below shows such a redundant inference and the corresponding resolution inference:

$$\frac{a \approx x \quad b \approx c}{a \approx b} \quad \text{and} \quad \frac{c \approx y \vee b \not\approx y \quad x \not\approx y \vee a \approx y}{b \not\approx y \vee a \approx y}.$$

Some resolution inferences correspond to unordered applications of paramodulation. For example, if we consider any reduction ordering in which $b \succ c$, then no ordered paramodulation is possible from clauses (2.1) and (2.2), but we have seen that the modified set of clauses allows for some resolution inferences. For quite some time people who worked on theorem proving using tableau and related methods tried to optimize the modification method so that less resolution inferences on the transformed set will correspond to redundant superposition inferences.

2.2. Constrained equality elimination

Such an optimized translation has been proposed by Bachmair et al. [1998] under the name of *constraint equality elimination*, or simply, *CEE-transformation*. This transformation comprises two of three parts. *Flattening* and *symmetry elimination* are the same as in Brand's transformation. The third part (transitivity elimination) is different from Brand's transformation in several aspects. To precise CEE-transformation we need some additional notions.

Constrained clauses.

- By an *ordering constraint*, or simply *constraint* we mean a conjunction of expressions which can be of two kinds:
 - *equality constraint* $s = t$, or
 - *ordering constraint* $s \succ t$ or $s \succeq t$,
 where s, t are terms.
- A substitution θ is called a *solution* to an equality constraint $s = t$ (respectively, ordering constraint $s \succ t$ or $s \succeq t$) if θ is grounding for $\text{vars}(s) \cup \text{vars}(t)$ and $s\theta = t\theta$ (respectively, $s\theta \succ t\theta$ or $s\theta \succeq t\theta$), where \succ is interpreted as a reduction ordering. A substitution θ is a solution to a conjunction C of constraints if θ is a solution to every equality or ordering constraint in C .
- A constraint C is *satisfiable* if it has a solution.
- Constraints C_1 and C_2 are called *equivalent* if they have the same sets of solutions.

Note that we use a different equality symbol in equality constraints, in order to emphasize that we mean the syntactic equality.

There is a vast literature on constraint solving for various orderings. For example, efficient methods for solving ordering constraints for lexicographic path orderings are described in [Nieuwenhuis 1993, Nieuwenhuis and Rubio 1995].

- We call a *constrained clause* a pair consisting of a clause C and a constraint C . Such a constrained clause is denoted by $C \cdot C$. We identify a constrained clause $C \cdot \top$, where \top is the empty conjunction, with the unconstrained clause C .

The logical semantics of the constraint clauses is defined through the notion of its ground instances defined below. Intuitively, a constraint clause represents the set of its ground instances.

- A *ground instance* of a constrained clause $C \cdot C$ is any ground clause $C\theta$ such that the substitution θ is a solution to the constraint C .
- We call two constrained clauses $C_1 \cdot C_1$ and $C_2 \cdot C_2$ *equivalent*, if they have the same ground instances. A set N of constrained clauses is *satisfiable* if the set of all its ground instances is satisfiable.

Transitivity elimination. One of the ideas of transitivity elimination in constraint equality elimination is to introduce the so-called *link constraints* into transitivity elimination. For example, the equation $s \approx t$ is transformed into $(t \not\approx x \vee s \approx x) \cdot (t \succeq x \wedge s \succ x)$. The names comes from the intuitive idea that the variable x is the *link variable* used to simulate transitivity.

The rationale behind introducing link constraints is that the sequence of equational replacements

$$s \approx s_0 \approx s_1 \approx \dots \approx s_n \approx t$$

(using equations $s_i \approx s_{i+1}$) can be simulated by a sequence of resolution inferences using $s_i \approx x_i \leftrightarrow s_{i+1} \approx x_i$, plus a final resolution step with the reflexivity axiom $x \approx x$ that instantiates the link variables. The ordering constraints

1. ensure that the variables x_i can only be *instantiated by minimal terms* among the s_i ;
 2. *block the search for alternative equational proofs* that apply the same equations but differ in the instantiation of the link variables.
- Formally, *transitivity elimination* is the following transformation:

$$\begin{aligned} s \approx t &\Rightarrow (t \not\approx x \vee s \approx x) \cdot (t \succeq x \wedge s \succ x) \\ s \not\approx t &\Rightarrow (t \not\approx x \vee s \not\approx x) \cdot (t \succeq x \wedge s \succeq x) \\ s \approx x &\Rightarrow (s \approx x) \cdot (s \succ x) \\ s \not\approx x &\Rightarrow (s \not\approx x) \cdot (s \succeq x) \end{aligned}$$

where t is not a variable. This transformation replaces every literal in the clause by the corresponding literals shown after \Rightarrow , and adds to the clause the corresponding constraints.

- The variables x introduced by the transformation are called the *link variables*.
- Each constraint introduced by the transformation (for example $s \succeq x$) is called the *link constraint*.

Consider again clauses (2.1) and (2.2). We have already considered how Brand's transformation behaves on these clauses. Now we apply CEE-transformation to these clauses, using any ordering in which c is the least element. With such ordering, no ordered paramodulation is possible. CEE-transformation yields four clauses:

$$\begin{aligned} & (b \not\approx y \vee c \approx y) \cdot (b \succeq y \wedge c \succ y) \\ & (c \not\approx y \vee b \approx y) \cdot (c \succeq y \wedge b \succ y) \\ & (c \not\approx y \vee f(y, x) \not\approx x) \cdot (c \succeq y \wedge f(y, x) \succeq x) \\ & x \approx x \end{aligned}$$

Note that the constraint $c \succ y$ is unsatisfiable and the constraint $c \succeq y$ can only be satisfied when $c = y$. The constraint $f(y, x) \succeq x$ is valid. Thus, we can simplify the set to

$$\begin{aligned} & c \not\approx c \vee b \approx c \\ & c \not\approx c \vee f(c, x) \not\approx x \\ & x \approx x \end{aligned}$$

Since the set contains $x \approx x$, both occurrences of $c \not\approx c$ can be removed, yielding

$$\begin{aligned} & b \approx c \\ & f(c, x) \not\approx x \\ & x \approx x \end{aligned}$$

No derivation by either resolution or model elimination is possible! This example shows some advantages of CEE-transformation compared to Brand's transformation.

The main property of CEE-transformation is preservation of satisfiability proved in [Bachmair et al. 1998]:

2.4. THEOREM. *A set N of unconstrained equational clauses is equationally unsatisfiable if and only if the transformed set is unsatisfiable.*

CEE-transformation can be used by any non-equational prover by adding constraints. The transformation without constraints is also sound and complete. The *least constant optimization* that removed the clause containing $c \succ y$ and simplified the clauses containing $c \succeq y$ in the example, can be also be used by provers that do not handle constraints. In fact, this transformation without constraints was described by Moser and Steinbach [1997] and used in some versions of the theorem prover SETHEO [Ibens and Letz 1997].

Compared to Brand's transformation, CEE-transformation has the following advantages:

1. the link constraints are added;
2. positive equations $t \approx x$ are not split into $x \not\approx y \vee t \approx y$.

The disadvantage is that negative equations $t \not\approx s$ are split giving longer clauses. Preliminary experiments with CEE-transformation [Bachmair et al. 1998] with the Protein prover [Baumgartner and Furbach 1994] on certain simple problems in group theory have shown that constraints may prohibit more than 99% of model elimination derivations even for comparatively simple examples.

Following [Bachmair et al. 1998], we give examples that show that Brand's transformation *cannot be improved* by either link constraints or non-splitting of positive literals $t \approx x$.

1. Suppose that Brand's transitivity elimination is equipped by link constraints. Consider an equationally unsatisfiable set of unit clauses $a \approx b$, $a \approx c$ and $b \not\approx c$. We obtain the following set of constrained clauses

$$\begin{aligned}
 &(b \not\approx x \vee a \approx x) \cdot (b \succeq x \wedge a \succ x) \\
 &(a \not\approx x \vee b \approx x) \cdot (a \succeq x \wedge b \succ x) \\
 &(c \not\approx x \vee a \approx x) \cdot (c \succeq x \wedge a \succ x) \\
 &(a \not\approx x \vee c \approx x) \cdot (a \succeq x \wedge c \succ x) \\
 &b \not\approx c \\
 &x \approx x
 \end{aligned}$$

We show that this set of constraint clauses is satisfiable given an ordering in which $c \succ b \succ a$. Indeed, the first and third clause contain the unsatisfiable constraint $a \succ x$ and hence can be removed. The remaining clauses are satisfiable even without the constraints. In short, the link constraints are not compatible with Brand's original transformations.

2. The set of three unit clauses $f(x) \approx x$, $g(x) \approx x$ and $f(x) \not\approx g(x)$ is equationally unsatisfiable. Suppose that we use Brand's transformation, but do not split the positive equations $t \approx x$. Then we obtain a satisfiable set of clauses

$$\begin{array}{ll}
 f(x) \approx x & f(x) \not\approx y \vee x \approx y \\
 g(x) \approx x & g(x) \not\approx y \vee x \approx y \\
 f(x) \not\approx g(x) & x \approx x
 \end{array}$$

In short, non-splitting of positive equations is not compatible with Brand's original transformations. Equally, we can say that constraint equality elimination cannot be improved by non-splitting negative literals $s \not\approx t$, even if we drop the link constraints.

Although the modification method has been proposed in the resolution framework, it had not been widespread used. The principal influence of this method on the equality reasoning was the proof of completeness of paramodulation without functional reflexivity axioms (conjectured by Robinson and Wos [1969]). The main tool for handling equality in resolution-based systems is paramodulation augmented with ordering restrictions and redundancy deletion. At the same time, most existing implementations of sequent-based methods with equality are based on the Brand's translation because the known ways of handling equality in resolution had not been generalized to sequent-based methods until very recently.

The distinction between CEE-transformation and Brand's transformation reflects the progress in equality handling made for more than twenty years (from 1975).

3. Free variable systems

The system $\mathbf{LK}^=$, has many nice proof-theoretic properties, but it is not very suitable for automatic proof-search. The main problem lies in the γ -rules, for example

$$\frac{\Gamma, F \varphi[x \mapsto t], F \exists x \varphi}{\Gamma, F \exists x \varphi} (\gamma)$$

where t can be *any term*. Application of such rules from conclusions to premises create too high nondeterminism. To make $\mathbf{LK}^=$ suitable for automatic proof-search, we have to introduce its *free variable* version, where instead of t we substitute a variable, which during the proof-search can be instantiated to some particular term t . In this section we consider two possible ways of making $\mathbf{LK}^=$ a free variable system. One gives rise to the semantic tableaux and the other to the inverse method. We also briefly consider the method of matings.

3.1. Free variable tableaux

For the formalization of free variable tableaux it is not enough to give sequent-style inference rules, like for $\mathbf{LK}^=$. Free variables are global for the whole tableau that represents a derivation. Thus, we need to introduce a calculus dealing with derivations. There are several ways of defining such calculi. One way is to define a calculus on trees, like in [Fitting 1990]. Another possibility is to only deal with the multiset of branches of such trees, which gives a more succinct formalization of inference rules (e.g. [Moser, Lynch and Steinbach 1995, Plaisted 1995, Degtyarev and Voronkov 1996b]). In this section we will present such a formalization for free variable tableaux. *Since we restrict ourselves to skolemized formulas, we do not need to consider δ -rules.*

- A *branch* is any finite multiset of signed formulas.
- A *tableau* is any finite multiset $\{\Gamma_1, \dots, \Gamma_n\}$ of branches, denoted by $\Gamma_1 \mid \dots \mid \Gamma_n$.
- The tableau with $n = 0$ is called the *empty tableau* and denoted by $\#$.

If Γ is a branch and $X \varphi$ is a signed formula, where $X \in \{T, F\}$, then $\Gamma, X \varphi$ denotes the branch $\Gamma \cup \{X \varphi\}$, and similar for $X \varphi, \Gamma$.

Let Γ be a branch in a tableau.

- The *multiset of literals on Γ* , denoted $lit(\Gamma)$, is defined by

$$lit(\Gamma) \doteq \{A \mid A \text{ is an atom and } T A \in \Gamma\} \cup \{\neg A \mid A \text{ is an atom and } F A \in \Gamma\}.$$

$$\begin{array}{c}
\frac{\Gamma_1, T \varphi \wedge \psi \mid \dots \mid \Gamma_n}{\Gamma_1, T \varphi, T \psi \mid \dots \mid \Gamma_n} (\alpha) \qquad \frac{\Gamma_1, F \varphi \wedge \psi \mid \dots \mid \Gamma_n}{\Gamma_1, F \varphi \mid \Gamma_1, F \psi \mid \dots \mid \Gamma_n} (\beta) \\
\\
\frac{\Gamma_1, T \varphi \vee \psi \mid \dots \mid \Gamma_n}{\Gamma_1, T \varphi \mid \Gamma_1, T \psi \mid \dots \mid \Gamma_n} (\beta) \qquad \frac{\Gamma_1, F \varphi \vee \psi \mid \dots \mid \Gamma_n}{\Gamma_1, F \varphi, F \psi \mid \dots \mid \Gamma_n} (\alpha) \\
\\
\frac{\Gamma_1, T \varphi \supset \psi \mid \dots \mid \Gamma_n}{\Gamma_1, F \varphi \mid \Gamma_1, T \psi \mid \dots \mid \Gamma_n} (\beta) \qquad \frac{\Gamma_1, F \varphi \supset \psi \mid \dots \mid \Gamma_n}{\Gamma_1, T \varphi, F \psi \mid \dots \mid \Gamma_n} (\alpha) \\
\\
\frac{\Gamma_1, T \neg \varphi \mid \dots \mid \Gamma_n}{\Gamma_1, F \varphi \mid \dots \mid \Gamma_n} (\neg) \qquad \frac{\Gamma_1, F \neg \varphi \mid \dots \mid \Gamma_n}{\Gamma_1, T \varphi \mid \dots \mid \Gamma_n} (\neg) \\
\\
\frac{T \forall x \varphi \mid \dots \mid \Gamma_n}{\Gamma_1, T \varphi[x \mapsto z], \Gamma_1, T \forall x \varphi, \mid \dots \mid \Gamma_n} (\gamma) \\
\\
\frac{F \exists x \varphi \mid \dots \mid \Gamma_n}{\Gamma_1, F \varphi[x \mapsto z], \Gamma_1, F \exists x \varphi, \mid \dots \mid \Gamma_n} (\gamma) \\
\\
\frac{\Gamma_1, T A, F B \mid \Gamma_2 \mid \dots \mid \Gamma_n}{(\Gamma_2 \mid \dots \mid \Gamma_n) \text{mgu}(A, B)} (abc)
\end{array}$$

In both rules (γ) the variable z does not occur in the premise. In the rule (abc) the formulas A and B are atomic formulas whose predicate symbol is different from \approx .

Figure 7: Calculus **TK**,

► A branch Γ is called *closed* if $\text{lit}(\Gamma)$ is inconsistent.

One can use any standard definition of inconsistency for first-order classical logic. In terms of the calculus $\mathbf{LK}^=$ the inconsistency means that the sequent $\text{lit}(\Gamma) \rightarrow$ is derivable in $\mathbf{LK}^=$. It is obvious that in a derivation of such a sequent one can only use the rules $(\neg \rightarrow)$, (Ax) , $(refl)$ and (\approx) . For logic without equality, the inconsistency is equivalent to the condition that $\text{lit}(\Gamma)$ contains a pair of complementary literals A and $\neg A$. For logic with equality, a criterion for inconsistency is given in Theorem 3.4 below.

► The *tableau calculus TK* is given in Figure 7 (compare it with $\mathbf{LK}^=$ in the form using signed formulas);

► α -, β -, γ -, and \neg -rules are called the *tableau expansion rules*;

► the rule (abc) is called the *atomic branch closure rule*.

Comparing the calculus **TK** introduced above with the calculus $\mathbf{LK}^=$ for signed formulas, we note the following. First, **TK** operates multisets of multisets of formulas (trees) and not multisets of formulas (branches) as $\mathbf{LK}^=$. So derivations in **TK** are always linear and not tree-like as in $\mathbf{LK}^=$. Second, applications of **TK**-inference rules correspond to counterapplications (from the conclusion to the premises) of

$LK^=$ -inference rules.

The following version of Herbrand theorem forms the theoretical basis for the tableau method:

3.1. THEOREM. *Let φ be a closed formula. It is provable in classical logic if and only if there is a tableau \mathcal{T} obtained from $F\varphi$ by applications of tableau expansion rules and a substitution θ such that any branch in $\mathcal{T}\theta$ is closed.*

Theorem 3.1 suggests the following way of proving φ : starting with $F\varphi$, we construct a tableau by some applications of the tableau expansion rules. Then, we try to find the substitution θ that makes all branches in the tableau closed. In the case without equality, this theorem and standard lifting yield the following theorem:

3.2. THEOREM. *Let ξ be a closed formula without equality. It is provable in classical logic if and only if there is a derivation in TK of the empty tableau $\#$ from the tableau $F\xi$.*

For logic with equality the situation is more complicated because there is no simple criterion of inconsistency of a branch.

Theorem 3.1 suggests the following notion (coming back to Lee and Chang [1973]).

- ▶ A branch Γ is called *substitutively inconsistent* if there is a substitution σ such that $lit(\Gamma)\sigma$ is inconsistent.
- ▶ A tableau $\Gamma_1 \mid \dots \mid \Gamma_n$ is called *substitutively inconsistent* if there is a substitution σ such that for every $i \in \{1, \dots, n\}$ the multiset $lit(\Gamma_i)\sigma$ is inconsistent.

In order to illustrate Theorem 3.1 for logic with equality, we come back to Example 1.3.

Now, we represent derivations in TK as trees with signed formulas (i.e. as semantic tableaux in the common meaning).

3.3. EXAMPLE. Consider again the formula of Example 1.3. After some application of tableau expansion rules we obtain the free variable tableau shown in Figure 8. This tableau has two branches. The multisets of literals on the two branches are, respectively

$$\begin{aligned} \{a \approx b, g(x, u, v) \not\approx g(y, fc, fd)\} \quad \text{and} \\ \{c \approx d, g(u, x, y) \not\approx g(v, fa, fb)\}. \end{aligned} \tag{3.1}$$

The tableau is substitutively inconsistent: the substitution $[x \mapsto fa, y \mapsto fb, u \mapsto fc, v \mapsto fd]$ makes both branches closed.

To demonstrate that a branch is closed, we have to be able to check a multiset of literals \mathcal{L} for inconsistency. To give a characterization of inconsistency for first-order logic with equality, we introduce some definitions. Let E be a multiset of equations and L_1, L_2 be terms or literals.

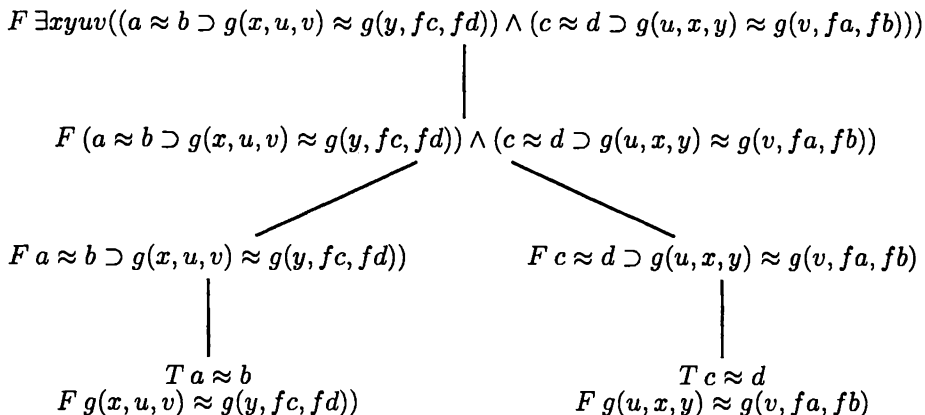


Figure 8: A free variable tableau derivation

- We write $L_1 \leftrightarrow_E L_2$ if there exists an equation $(s \approx t) \in E$ such that L_1 is $L[s]$ and L_2 is $L[t]$. We denote by \leftrightarrow_E^* the reflexive and transitive closure of \leftrightarrow_E . The following statement characterizes inconsistency of a multiset of literals:

3.4. THEOREM. *Let \mathcal{L} be a multiset of literals and $\mathcal{E}(\mathcal{L})$ be the multiset of equations in \mathcal{L} . Then the following conditions are equivalent:*

1. \mathcal{L} is inconsistent;
2. either there is a literal $(s \not\approx t) \in \mathcal{L}$ such that $s \leftrightarrow_{\mathcal{E}(\mathcal{L})}^* t$, or there are non-equality literals $A, \neg B \in \mathcal{L}$ such that $A \leftrightarrow_{\mathcal{E}(\mathcal{L})}^* B$.

The inconsistency of a multiset of literals can be checked using any congruence closure algorithm [e.g. Shostak 1978]. However, the real question is how to close tableau branches for a given free variable tableau. This question leads to simultaneous rigid E -unification considered in Section 5.

3.2. Matings

In this section we briefly discuss the method of matings which can be considered as another formalization of proof-search in a free-variable sequent calculus.

Let us come back to the formula of Example 1.3. The tableau constructed in Example 3.3 can be represented as a matrix in the method of matings as shown in Figure 9. For a better illustration, we also show the construction of the matrix. After translation to the negation normal form, the negation of the input formula becomes $\forall xyuv((a \approx b \wedge g(x, u, v) \not\approx g(y, fc, fd)) \vee (c \approx d \wedge g(u, x, y) \not\approx g(v, fa, fb)))$. From

$$\begin{aligned}
& \forall xyuv((a \approx b \wedge g(x, u, v) \not\approx g(y, fc, fd)) \vee (c \approx d \wedge g(u, x, y) \not\approx g(v, fa, fb))) \Rightarrow \\
& (a \approx b \wedge g(x, u, v) \not\approx g(y, fc, fd)) \vee (c \approx d \wedge g(u, x, y) \not\approx g(v, fa, fb)) \Rightarrow \\
& \left[a \approx b \wedge g(x, u, v) \not\approx g(y, fc, fd) \quad c \approx d \wedge g(u, x, y) \not\approx g(v, fa, fb) \right] \Rightarrow \\
& \left[\left[\begin{array}{c} a \approx b \\ g(x, u, v) \not\approx g(y, fc, fd) \end{array} \right] \quad c \approx d \wedge g(u, x, y) \not\approx g(v, fa, fb) \right] \Rightarrow \\
& \left[\left[\begin{array}{c} a \approx b \\ g(x, u, v) \not\approx g(y, fc, fd) \end{array} \right] \quad \left[\begin{array}{c} c \approx d \\ g(u, x, y) \not\approx g(v, fa, fb) \end{array} \right] \right]
\end{aligned}$$

Figure 9: Matings

this formula, we can perform the sequence of steps shown in Figure 9. The matrix in the figure has two vertical paths which are precisely the multisets in (3.1).

In general, for every matrix M consisting only of literals we can construct a tableau \mathcal{T} such that the multiset of vertical paths in M is the multiset of multisets of literals on the branches of \mathcal{T} . Methods and algorithms described in this chapter work on tableaux represented as multisets of branches. To modify them for the connection format, one has to imagine matrices as multisets of branches³.

A detailed explanation of equational matings can be found in [Gallier, Raatz and Snyder 1989, Gallier,arendran, Raatz and Snyder 1992, Gallier 1992].

3.3. The inverse method

Derivations in the free variable tableau system correspond to the *backward* (i.e. from conclusions to premises) proof-search in $\mathbf{LK}^=$. There is an orthogonal method of *forward* proof-search in $\mathbf{LK}^=$ which gives rise to the inverse method. Forward proof-search starts with axioms and tries to derive the goal formula. The whole idea of the forward direction of proof-search seems at the first sight strange. But in the case of sequent calculi, the possibility of such proof-search direction is based on some proof-theoretic properties of $\mathbf{LK}^=$, most notably

- the *subformula property*: for every derivation Π of a formula ξ , all sequents occurring in Π consist of formulas of the form $\varphi\theta$, where φ is a subformula of ξ .

³Some rules formulated below, for example tableau paramodulation or superposition, have a convenient presentation on tableaux but have no straightforward analogue for matrices.

This property together with the use of free variables gives rise to a proof-search method originally described in [Maslov 1964, Maslov 1983a, Maslov 1987]. Since the inverse method is much less known than the tableau method, we describe in this section a version of the inverse method for logic without equality.

Our formalization of the inverse method will mainly follow [Voronkov 1992]. Other recent formalizations may be found in [Lifschitz 1989, Tammet 1996, Voronkov 2000b, Voronkov 2001a] and in [Degtyarev and Voronkov 2001b] (Chapter 4 of this Handbook). In this section, we only formalize the inverse method for logic without equality. The inverse method for logic with equality is considered in Section 8.3. Given a closed formula ξ as the goal, the inverse method constructs a logical system \mathbf{IK}_ξ which is a specialization of $\mathbf{LK}^=$ intended for proving ξ . This calculus works on sequents that consist of formulas $\varphi\theta$ such that φ is a subformula of ξ .

For simplicity, we assume that ξ is in the skolemized negation normal form and give a calculus consisting of the rules (\vee) , (\wedge) and (\exists) , like the calculus of Figure 6. Before defining \mathbf{IK}_ξ , we define some operations related to substitutions.

Let s, t be terms.

- ▶ A *weak most general unifier* of s, t , denoted $\text{wmgu}(s, t)$, is a pair of substitutions $(\rho\sigma, \sigma)$ such that ρ is a renaming substitution, $s\rho$ and t have no common variables, and σ is a most general unifier of $s\rho$ and t .
- ▶ A *most general unifier* of substitutions σ_1, σ_2 , denoted $\text{mgu}(\sigma_1, \sigma_2)$, is a substitution θ defined as follows. Let $\{x_1, \dots, x_n\} = \text{dom}(\sigma_1) \cup \text{dom}(\sigma_2)$. Then θ is any simultaneous most general unifier of tuples $\langle x_1\sigma_1, \dots, x_n\sigma_1 \rangle$ and $\langle x_1\sigma_2, \dots, x_n\sigma_2 \rangle$.
- ▶ We denote by σ_{-x} the substitution defined as follows. For every variable y ,

$$y\sigma_{-x} = \begin{cases} x, & \text{if } x = y; \\ y\sigma, & \text{otherwise} \end{cases}$$

Using these notions, we define the calculus \mathbf{IK}_ξ formalizing the inverse method in the following way. Note that we assume that ξ is in Skolem negation normal form, and therefore contains no universal quantifiers.

- ▶ *Sequents in the calculus \mathbf{IK}_ξ* consist of formulas $\varphi\theta$ such that φ is a subformula of ξ and all variables in $\text{dom}(\theta)$ are free variables of φ .
- ▶ The *calculus \mathbf{IK}_ξ* is given in Figure 10.

In the rules of this calculus we assume all premises to be variable-disjoint. This can be achieved by adding an implicit variable renaming rule. A more detailed presentation can be found in [Degtyarev and Voronkov 2001b] (Chapter 4 of this Handbook).

Consider an example inverse method proof of the formula $\exists x\exists y\forall z(P(x) \vee P(y) \supset P(z))$. After skolemization and translation into the negation normal form we obtain the formula $\xi = \exists x\exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x, y)))$. An \mathbf{IK}_ξ -derivation of ξ is given in Figure 11.

As related to proof-search, there are several terms to distinguish the tableau method from the inverse method:

$$\begin{array}{c}
\frac{}{A\sigma_1, \neg B\sigma_2} (Ax) \qquad \frac{\Gamma, \varphi\sigma}{\Gamma, (\varphi \vee \psi)\sigma} (\vee) \qquad \frac{\Gamma, \psi\sigma}{\Gamma, (\varphi \vee \psi)\sigma} (\vee) \\
\\
\frac{\Gamma, \varphi\sigma_1 \quad \Delta, \psi\sigma_2}{(\Gamma, \Delta, \varphi\sigma_1 \wedge \psi\sigma_2)\text{mgu}(\sigma_1, \sigma_2)} (\wedge) \qquad \frac{\Gamma, \varphi\sigma}{\Gamma, (\exists x\varphi)\sigma_{-x}} (\exists) \\
\\
\frac{\Gamma, \varphi\sigma_1, \varphi\sigma_2}{(\Gamma, \varphi\sigma_1)\text{mgu}(\sigma_1, \sigma_2)} (fac)
\end{array}$$

In the rule (Ax) , A and $\neg B$ are subformulas of ξ and $(\sigma_1, \sigma_2) = \text{wmgu}A, B$. In the other rules, the formulas $\varphi \wedge \psi$, $\varphi \vee \psi$ and $\exists x\varphi$ are subformulas of ξ .

Figure 10: Calculus \mathbf{IK}_ξ

$$\begin{array}{c}
\frac{}{\neg P(x)[x \mapsto f(x, y)], P(f(x, y))} (Ax) \quad \frac{}{\neg P(y)[y \mapsto f(x, y)], P(f(x, y))} (Ax) \\
\frac{}{(\neg P(x) \wedge \neg P(y))[y \mapsto f(x, y)], P(f(x, y)), P(f(x, y))} (\wedge) \\
\frac{}{((\neg P(x) \wedge \neg P(y)) \vee P(f(x, y)))[x \mapsto f(x, y), y \mapsto f(x, y)], P(f(x, y)), P(f(x, y))} (\vee) \\
\frac{}{\exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x, y))), P(f(x, y)), P(f(x, y))} (\exists) \\
\frac{}{\exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x, y))), P(f(x, y))} (fac) \\
\frac{}{\exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x, y))), ((\neg P(x) \wedge \neg P(y)) \vee P(f(x, y)))} (\vee) \\
\frac{}{\exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x, y))), \exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x, y)))} (\exists) \\
\frac{}{\exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x, y)))} (fac)
\end{array}$$

Figure 11: An \mathbf{IK}_ξ -derivation

Tableau method	Inverse method	
bottom-up	top-down	(proof-theoretic terminology)
top-down	bottom-up	(search-related terminology)
backward	forward	
direct	indirect	
goal-oriented	inverse	
analytic	non-analytic	
nonlocal	local	

The inverse method for classical logic can be simulated by hyperresolution using a structure-preserving clause-form translation that has been described in various forms in [Maslov 1983b, Eder 1984, Plaisted and Greenbaum 1986, Boy de la Tour 1990]. Variants of this translation can be found in [Nonnengart and Weidenbach 2001, Weidenbach 2001, Degtyarev and Voronkov 2001b] (Chapters 6, 27, and 4 of

this Handbook). We consider an example in Section 8.3.

4. Early history

Cut-free sequent calculi for first-order logic introduced by Gentzen [1934] turned out to be an important tool for investigating basic proof-theoretic problems (e.g. Gentzen [1936] and Girard [1987b]). It was also realized that sequent systems are a convenient tool for designing proof-search algorithms.

One of the first sequent-based proof procedures has been proposed by Kanger [1957] independently of the more known papers [Beth 1983, Beth 1959] (the monograph [Kanger 1957] is based on lectures made in Stockholm University in 1955). The distinctive features of the calculi used in [Kanger 1957, Kanger 1963] are the absence of structure rules and invertibility of all inference rules (calculi with similar properties have been studied in [Matulis 1962, Matulis 1963] and [Curry 1963]). Antecedents and succedents of sequents in Kanger's calculi are multisets of formulas. Invertibility of rules allowed Kanger to prove completeness "by means of arguments which are new in some respect and which involve a new turn to the notion of validity". This means that if a sequent S is unprovable, then *any* derivation tree for S has a branch containing a countermodel for S .

Search for a countermodel is the basis for the tableau method developed by [Beth 1983, Beth 1959]. [Beth 1959] noted that his tableau calculus F can equivalently be described as a sequent calculus similar to Kleene's [1952] $G3$. In fact, the tableau calculus of Beth was even more similar to the calculus of Kanger [1957].

As far as we know, the first sequent-based proof system for logic with equality has been proposed and *implemented* by Wang [1960]. Equality has been axiomatized by one axiom

$$\Gamma \rightarrow \Delta, t \approx t$$

and one inference rule

$$\frac{\Gamma, t \approx s \rightarrow \Delta[x \mapsto s]}{\Gamma, t \approx s \rightarrow \Delta[x \mapsto t]}$$

The first extensive analysis of sequent calculi for equational logic have been made in [Kanger 1963] and in the papers of the Leningrad group of mathematical logic [Rogava 1967, Lifschitz 1967, Lifschitz 1968, Orevkov 1969, Pliuškevičiene 1969]. The main rule for handling equality used by Kanger is based on the simultaneous replacement of all occurrences of the same subterm:

$$\frac{s \approx t, \Gamma_t^s \rightarrow \Delta_t^s}{s \approx t, \Gamma \rightarrow \Delta},$$

where Γ_t^s denotes the result of the simultaneous replacement of all occurrences of the term s by t in Γ . This rule is close to *simultaneous paramodulation* of Benanav

[1990]. It is notable that the system with simultaneous paramodulation has the lifting property.

Kanger [1963] also introduced some restrictions on derivations that are interesting from the viewpoint of the current knowledge of the area. First, he only allowed *nonincreasing* applications of the above equality rule, i.e. applications in which the depth of t is not greater than the depth of s . Second, he restricted himself to *regular derivations* only.

Lifschitz [1967, 1968] and Orevkov [1969] considered some extravagant ways of orientation of equations. For example, Orevkov [1969] oriented equations using *any* asymmetric binary relation R . The replacement of s by t using $s \approx t$ has only been allowed when it is not true that sRt .

In addition to the brilliant⁴ formalization of equality, the free variable system of Kanger used a new strategy for instantiating variables in the applications of γ -rules (i.e. the rules $(\rightarrow \exists)$ and $(\forall \rightarrow)$). His strategy of instantiating variables is based on two ideas: the use of free variables and instantiation of free variables by terms already occurring in the derivation.

Free variables used by Kanger have been originally introduced by Prawitz [1983] for logic without equality. The idea is to introduce a new kind of variables (called “*dummies*” by both Prawitz [1983] and Kanger [1963]), and to delay instantiation of these variables until necessary information for it is obtained. Comparing this approach to an earlier work of Beth [1959], Prawitz [1983] noted: “the solution proposed here is quite different but well-suited for mechanical use”. This method was independently proposed in Russia by N. Shanin in 1962 [see Maslov 1964] and has been characterized as the “*metavariation method*” in [Maslov, Mints and Orevkov 1983]. Information for instantiation is provided by constructing an uninstantiated proof, and checking from time to time whether one can find values for dummies which make it a valid proof. In Kanger [1963] this check is reduced to verifying that the top sequents are “directly demonstrable”, i.e. can be obtained from axioms by applications of equality rules. The possibility of applying all equality rules before all other rules has also been demonstrated in [Orevkov 1969] and used in [Maslov 1971].

Dummies or metavariables have later been called “*free variables*” in [Fitting 1990]. For instantiation of dummies, or free variables, Kanger [1963] proposed an approach later called *minus-normalization* in [Maslov and Mints 1983]. According to this approach, instantiation of variables in the applications of γ -rules from the conclusion to the premise is made only by ground terms explicitly occurring in the conclusion. This method is complete for first-order logic. However, for a language with function symbols minus-normalization is interesting mostly theoretically. Even in simplest cases, minus-normalization requires a huge number of instantiations. For example, in the tableau of Example 1.3, we have to consider 8^4 possible instantiations of variables x, y, u, v by the terms in the set $\{a, b, c, d, fa, fb, fc, fd\}$. Moreover, the use of minus-normalization can lead to considerable growth of derivations. Some results on minus-normalization are proved in [Norgela 1974].

⁴Brilliant, since Kanger has anticipated many tendencies used in the modern methods of handling equality in automated deduction.

In the modern terminology, we can say that minus-normalization gives an incomplete (but terminating) algorithm for simultaneous rigid E -unification discussed in Section 5.

For free variable tableaux, more practical way of instantiating variables is the introduction of unification in inference rules, for example branch closure rules considered in subsequent sections. Fitting [1994] writes

It occurred independently to several people that free variables could be used instead, in a way that gave an important role to unification.

Results considered in this section provided the proof-theoretic basis for further research in sequent calculi with equality. Later research introduced methods based on simultaneous rigid E -unification and variants of rigid paramodulation considered in Sections 6 and 7. An overview of novelties introduced in automated deduction by Kanger can be found in [Degtyarev and Voronkov 2001c].

5. Simultaneous rigid E -unification

In Section 3 we considered the free variable version TK of semantic tableaux (in fact, without equality). In this section we consider the problem of instantiating free variable semantic tableaux with equality which gives rise to *simultaneous rigid E -unification* introduced by Gallier et al. [1987]. In Section 5.1 we define simultaneous rigid E -unification and show how it arises in semantic tableaux with equality. In Section 5.2 we consider some properties of simultaneous rigid E -unification.

5.1. Definition and examples

The actual problem in the tableau-based theorem proving with equality, as established by Theorem 3.1, is the problem of *finding a substitution* that makes all tableau branches closed. The problem of finding such a substitution, together with Theorem 3.4, leads to *simultaneous rigid E -unification*. Simultaneous rigid E -unification arises after we fixed, according to Theorem 3.4, a literal $s \not\approx t$ or two literals $A, \neg B$ on every branch of a tableau.

Let us give formal definitions.

- A *rigid equation* is an expression of the form $E \vdash_{\forall} s \approx t$, where E is a finite set of equations and s, t are terms. The set E is called *the left-hand side* of this rigid equation, and $s \approx t$ is called its *right-hand side*.
- A *solution* to such a rigid equation is any substitution θ such that $E\theta \vdash s\theta \approx t\theta$. A rigid equation is *solvable* if it has a solution.
- The *rigid E -unification* problem is the problem of determining whether a given rigid equation is solvable.

In other words, θ is a solution to a rigid equation $E \vdash_{\forall} s \approx t$ if and only if the set $E\theta \cup \{s\theta \not\approx t\theta\}$ is inconsistent.

- A *system of rigid equations* is any finite set of rigid equations;

- A *solution* to a system R of rigid equations is any substitution which is a solution to every rigid equation in R ;
- The *simultaneous rigid E-unification* problem is the problem of determining whether a given system of rigid equations possesses a solution.

In Example 3.3, to close all branches in the tableau, we had to find a substitution θ such that the following two multisets of literals are inconsistent:

$$\begin{aligned} \{a \approx b, g(x, u, v)\theta \not\approx g(y, fc, fd)\theta\}, \\ \{c \approx d, g(u, x, y)\theta \not\approx g(v, fa, fb)\theta\}. \end{aligned} \quad (5.1)$$

This (simultaneous) inconsistency problem can be expressed through the system of two rigid equations:

$$\begin{aligned} a \approx b \vdash_{\forall} g(x, u, v) \approx g(y, fc, fd), \\ c \approx d \vdash_{\forall} g(u, x, y) \approx g(v, fa, fb). \end{aligned} \quad (5.2)$$

This system of rigid equations has one solution $[x \mapsto fa, y \mapsto fb, u \mapsto fc, v \mapsto fd]$. In order to check this, we should verify

$$\begin{aligned} a \approx b \vdash g(fa, fc, fd) \approx g(fb, fc, fd), \\ c \approx d \vdash g(fc, fa, fb) \approx g(fd, fa, fb). \end{aligned}$$

In general, there can be more than one system of rigid equations, each of whom expressing that all branches in a tableau are closed. For example, for the tableau

$$T a \approx b, F b \approx c, F a \approx x \mid T P(x), F P(b)$$

we can construct two systems of rigid equations expressing its inconsistency:

$$\begin{array}{ll} a \approx b \vdash_{\forall} b \approx c & \text{and} \quad a \approx b \vdash_{\forall} a \approx x \\ \vdash_{\forall} x \approx b & \vdash_{\forall} x \approx b \end{array}$$

Simultaneous rigid *E*-unification was introduced by Gallier et al. [1987] who hoped that the problem is decidable. The terminology “simultaneous rigid *E*-unification” can be explained as follows. The word “rigid” is introduced to distinguish rigid *E*-unification from *E*-unification. The latter problem can be formulated as follows. Given a (finite) set of equations $E = \{s_1 \approx t_1, \dots, s_n \approx t_n\}$ and the equation $s \approx t$, find a substitution θ such that $\forall(s_1 \approx t_1), \dots, \forall(s_n \approx t_n) \vdash s\theta \approx t\theta$. For rigid *E*-unification, we try to find a substitution θ such that $\vdash \forall(s_1\theta \approx t_1\theta \wedge \dots \wedge s_n\theta \approx t_n\theta \supset s\theta \approx t\theta)$.

In *E*-unification, all variables in the equations $s_i \approx t_i$ are treated as universal, while all variables in rigid equations are *rigid*: if we substitute a term for a variable in any part of a rigid equation, we must substitute it in the whole rigid equation. The word “simultaneous” means that we have to find a simultaneous solution to several rigid equations.

Consider an example rigid equation $x \cdot y \approx y \cdot x \vdash_{\forall} a \cdot (b \cdot a) \approx (a \cdot b) \cdot a$. It has no solutions. However, we have $\forall x \forall y (x \cdot y \approx y \cdot x) \vdash a \cdot (b \cdot a) \approx (a \cdot b) \cdot a$, and hence the terms $a \cdot (b \cdot a)$ and $(a \cdot b) \cdot a$ are E -unifiable with respect to the equality theory $E = \{x \cdot y \approx y \cdot x\}$.

Surprisingly, rigid variables in the context of resolution theorem proving have been used much earlier by Chang [1972] and Lee and Chang [1973] in their *V-resolution* and *V-paramodulation* rules. Chang and Lee tried to use rigid variables in order to capture Prawitz's procedure by resolution and to formalize the idea of reasoning with bounded resources. Later, *V-paramodulation* have been described as *rigid paramodulation* by Plaisted [1995]. We consider resolution-based theorem proving with rigid variables in Section 7.3.

5.2. Undecidability and other properties of simultaneous rigid E -unification

Gallier et al. [1987] introduced simultaneous rigid E -unification and asked whether this problem is decidable. Several papers published in 1988–1992 [Gallier, Narendran, Plaisted and Snyder 1988, Gallier, Narendran, Plaisted and Snyder 1990, Gallier et al. 1992, Gallier 1992, Goubault 1994] described faulty proofs of the decidability. Finally, in 1995 the problem was proved to be undecidable by Degtyarev and Voronkov [1995e] by reduction of *monadic semi-unification* whose undecidability was proved by Baaz [1993]. More comprehensive proofs of the undecidability appeared

1. in [Degtyarev and Voronkov 1995d, Degtyarev and Voronkov 1996f] by reduction of second-order unification whose undecidability was proved by Goldfarb [1981];
2. in [Degtyarev and Voronkov 1996e] by reduction of tenth Hilbert's problem;
3. in [Veanes 1996, Veanes 1997a] by direct encoding of Turing machines.

We will briefly explain the ideas of the undecidability proof of [Degtyarev and Voronkov 1995d, Degtyarev and Voronkov 1996f] and consider further results on simultaneous rigid E -unification below.

This undecidability proof is also interesting because it gives an encoding of second-order unification by simultaneous rigid E -unification. We will present second-order unification here informally, for more detail see [Goldfarb 1981] or [Degtyarev and Voronkov 1995d, Degtyarev and Voronkov 1996f]. *Second-order unification* can be considered as a formalization of application of substitutions to terms.

Second-order unification (see [Dowek 2001], Chapter 16 of this Handbook) differs from ordinary unification by the use of second-order variables. A second-order variable x may occur in terms in the form $x(t_1, \dots, t_n)$. Second-order substitutions substitute for such variables λ -terms of the form $\lambda w_1 \dots w_n. t$, where t is any first-order term. The application of the substitution $[x \mapsto \lambda w_1 \dots w_n. t]$ to the term $x(t_1, \dots, t_n)$ yields the term $t[w_1 \mapsto t_1, \dots, w_n \mapsto t_n]$. Ordinary first-order variables can be considered as second-order variables of 0 arguments.

An example of a second-order unification problem is $f(z(y(a))) \approx z(y(b))$. By introduction of new variables, any such problem can be reduced to a set of equations of the form $x(t_1, \dots, t_n) \approx t$, where $n \geq 0$ and t, t_1, \dots, t_n are first-order terms. For

example, the second-order equation $f(z(y(a))) \approx z(y(b))$ can be reduced to the following set of equations:

$$\{y(a) \approx u_1, z(u_1) \approx u_2, y(b) \approx u_3, z(u_3) \approx f(u_2)\}.$$

Consider the following second-order equation

$$x(y, g(y)) \approx f(g(z), a, y). \quad (5.3)$$

We will be interested in its ground solutions, i.e. in solutions θ such that $x\theta, y\theta, z\theta$ are ground terms. We also assume that we are looking for its solutions in the signature $\{f, g, a, b\}$. Among ground solutions to this equation are the following:

$$[x \mapsto \lambda w_1 w_2. f(g(b), a, b), y \mapsto b, z \mapsto b], \quad (5.4)$$

$$[x \mapsto \lambda w_1 w_2. f(w_2, a, w_1), y \mapsto b, z \mapsto b], \quad (5.5)$$

$$[x \mapsto \lambda w_1 w_2. f(w_2, w_1, a), y \mapsto a, z \mapsto a], \quad (5.6)$$

$$[x \mapsto \lambda w_1 w_2. f(w_1, a, g(b)), y \mapsto g(b), z \mapsto b]. \quad (5.7)$$

In order to encode second-order unification, one can use the following properties of simultaneous rigid E -unification [Degtyarev and Voronkov 1995d, Degtyarev and Voronkov 1996f]. First, one can express the property of being a ground term of a given signature \mathcal{G} having at least one constant c . Let t be any term. Introduce the following rigid equation:

$$Gr(t, \mathcal{G}) \doteq \{f(c, \dots, c) \approx c \mid f \in \mathcal{G}\} \vdash_{\forall} t \approx c.$$

We have

5.1. LEMMA. *A substitution θ is a solution to $Gr(t, \mathcal{G})$ if and only if $t\theta$ is a ground term of the signature \mathcal{G} .*

In a similar way, one can represent all regular sets [see Goubault 1994, Plaisted 1995, Veanes 1996].

Second, simultaneous rigid E -unification can represent application of substitutions. We will temporarily use the substitution notation $[c_1 \mapsto t_1, \dots, c_n \mapsto t_n]$, where c_i are constants, to denote the operation of the simultaneous replacement of *all* occurrences of c_i by t_i . Representation of substitutions is based on the following property of equational logic:

5.2. LEMMA. *Let c_1, \dots, c_n be pairwise different constants and t_1, \dots, t_n be first-order terms such that c_i does not occur in t_j for all $i, j \in \{1, \dots, n\}$. Then $c_1 \approx t_1, \dots, c_n \approx t_n \vdash s_1 \approx s_2$ if and only if $s_1[c_1 \mapsto t_1, \dots, c_n \mapsto t_n] = s_2[c_1 \mapsto t_1, \dots, c_n \mapsto t_n]$.*

This lemma and Lemma 5.1 are enough to represent second-order unification. Indeed, consider any second-order equation of the form $x(t_1, \dots, t_n) \approx t$ in a signature \mathcal{G} , where t_1, \dots, t_n are first-order terms. Let c_1, \dots, c_n be new constants. Consider the following system R of rigid equations:

$$\begin{aligned} &Gr(t_1, \mathcal{G}), \\ &\dots \\ &Gr(t_n, \mathcal{G}), \\ &Gr(t, \mathcal{G}), \\ &Gr(x, \mathcal{G} \cup \{c_1, \dots, c_n\}), \\ &c_1 \approx t_1, \dots, c_n \approx t_n \vdash_{\forall} x \approx t. \end{aligned}$$

Let θ be any substitution. Consider when θ solves this system of rigid equations. By Lemma 5.1, all terms $t_1\theta, \dots, t_n\theta, t\theta$ are ground terms in the signature \mathcal{G} , and $x\theta$ is a ground term in $\mathcal{G} \cup \{c_1, \dots, c_n\}$. Since c_1, \dots, c_n do not occur in $t_1\theta, \dots, t_n\theta, t\theta$, we can apply Lemma 5.2 to the last rigid equation in R , obtaining

$$x\theta[c_1 \mapsto t_1\theta, \dots, c_n \mapsto t_n\theta] = t\theta.$$

Let $\theta = [x_1 \mapsto s_1, \dots, x_m \mapsto s_m, x \mapsto s]$. It is easy to see that θ is a solution to R if and only if the substitution $[x_1 \mapsto s_1, \dots, x_m \mapsto s_m, x \mapsto \lambda c_1 \dots c_n. s]$ is a solution to $x(t_1, \dots, t_n) \approx t$. Thus, we can encode second-order unification in simultaneous rigid E -unification in a natural way.

If we apply this encoding to second order equation (5.3), we obtain the following set of rigid equations:

$$\begin{aligned} &f(a, a, a) \approx a, g(a) \approx a, b \approx a \vdash_{\forall} y \approx a, \\ &f(a, a, a) \approx a, g(a) \approx a, b \approx a \vdash_{\forall} z \approx a, \\ &f(a, a, a) \approx a, g(a) \approx a, b \approx a, c_1 \approx a, c_2 \approx a \vdash_{\forall} x \approx a, \\ &c_1 \approx y, c_2 \approx g(y) \vdash_{\forall} x \approx f(g(z), a, y). \end{aligned}$$

Some solutions to this system of rigid equations are

$$\begin{aligned} &[x \mapsto f(g(b), a, b), y \mapsto b, z \mapsto b]; \\ &[x \mapsto f(c_2, a, c_1), y \mapsto b, z \mapsto b]; \\ &[x \mapsto f(c_2, c_1, a), y \mapsto a, z \mapsto a]; \\ &[x \mapsto f(c_1, a, g(b)), y \mapsto g(b), z \mapsto b]. \end{aligned}$$

The reader can compare them with solutions (5.4)–(5.7) of the original second-order equation. Other relations between second-order unification and simultaneous rigid E -unification are discussed by Veanes [1998a].

Although the undecidability result for simultaneous rigid E -unification seems to diminish the value of this notion, it has some other applications. Gallier et al. [1990] noted that “rigid E -unification and Girard’s linear logic [Girard 1987a] share

the same spirit". Further results concerning simultaneous rigid E -unification have shown its close connections to a remarkable number of problems in logic with equality. Let us briefly review some of these results.

The undecidability of simultaneous rigid E -unification implies the undecidability of the following problems:

1. (*Quantifier-free*) *formula instantiation*: given a quantifier-free formula $\varphi(\bar{x})$, do there exist terms \bar{t} such that $\varphi(\bar{t})$ is provable [Voronkov 1998a, Voronkov 1999].
2. \exists^* -*fragment of intuitionistic logic with equality*, and also the *prenex fragment of intuitionistic logic with equality* [Degtyarev and Voronkov 1996a]. This result is interesting because both these fragments for intuitionistic logic without equality are decidable and PSPACE-complete [Degtyarev and Voronkov 1996a, Voronkov 1996b, Voronkov 2001b].
3. The *Herbrand skeleton problem* (in the terminology of Voda and Komara [1995]) is the following series of decision problems, for every natural number n . Given a formula $\exists \bar{x} \varphi(\bar{x})$, where $\varphi(\bar{x})$ is quantifier-free, do there exist sequences of terms $\bar{t}_1, \dots, \bar{t}_n$ such that the formula $\varphi(\bar{t}_1) \vee \dots \vee \varphi(\bar{t}_n)$ is provable. This problem naturally arises from the Herbrand theorem: a formula $\exists \bar{x} \varphi(\bar{x})$ is provable if and only if there exists a natural number n and sequences of terms $\bar{t}_1, \dots, \bar{t}_n$ such that the formula $\varphi(\bar{t}_1) \vee \dots \vee \varphi(\bar{t}_n)$ is provable.
4. *Skeleton instantiation problem* (also considered in Section 9): given a formula and a derivation skeleton, is there a derivation of this formula with this skeleton.

and some similar problems [see Degtyarev, Gurevich and Voronkov 1996, Voronkov 1998d, Voronkov 1998a, Voronkov 1999, Degtyarev, Gurevich and Voronkov 2000].

The undecidability of simultaneous rigid E -unification posed a problem of its replacement by other techniques. These techniques will be considered in Sections 6 and 7. However, in some cases automated reasoning with simultaneous rigid E -unification is still possible, because of the decidability of some its fragments. In fact, the (un)decidability of simultaneous rigid E -unification depends on the signature \mathcal{F} . We formulate some results related to special cases of simultaneous rigid E -unification:

1. Gallier et al. [1988] proved the decidability and NP-completeness of rigid E -unification. Although it is an interesting result by itself, it does not contribute much to automated reasoning problems because practical problems usually give rise to the simultaneous case.
2. Degtyarev, Matiyasevich and Voronkov [1995] note that the function-free fragment of simultaneous rigid E -unification is NP-complete [see also Degtyarev, Matiyasevich and Voronkov 1996].
3. [Degtyarev, Matiyasevich and Voronkov 1995, Degtyarev, Matiyasevich and Voronkov 1996] prove the decidability of simultaneous rigid E -unification in the language with one unary function symbol and any number of constants (which implies the decidability of all problems mentioned above for this language). The complexity of this fragment is unknown since the proof uses a reduction to the Diophantine problem for addition and divisibility [see Beltyukov 1976, Mart'janov 1977, Lipshitz 1978] whose complexity is now known [Lipshitz 1981].

4. The decidability of simultaneous rigid E -unification in the signature \mathcal{F} with only unary function symbols is an open problem. It is known [Degtyarev, Matiyasevich and Voronkov 1995, Degtyarev, Matiyasevich and Voronkov 1996] that it is decidable in any such signature if and only if it is decidable in the signature with two unary function symbols and any number of constants. In addition, the word equation problem [see Makanin 1977] has a simple reduction to simultaneous rigid E -unification in such signatures. (It is easy to show such a reduction by modifying the encoding of second-order unification considered above to the case of unary function symbols.) [Gurevich and Voronkov 1997a, Gurevich and Voronkov 1997b, Gurevich and Voronkov 1999] proved that simultaneous rigid E -unification in the case of unary function symbols is equivalent to an extension of word equations.
5. Simultaneous rigid E -unification with ground left-hand sides is undecidable [Plaisted 1995].
6. [Veanes 1997a, Veanes 1997b] improved this results by showing that simultaneous rigid E -unification is undecidable even in the case of one binary function symbol, one constant, two variables and ground left-hand sides.
7. [Degtyarev, Gurevich, Narendran, Veanes and Voronkov 1997, Degtyarev, Gurevich, Narendran, Veanes and Voronkov 1998, Degtyarev, Gurevich, Narendran, Veanes and Voronkov 2000] proved that simultaneous rigid E -unification with one variable is decidable.

We do not give details and techniques used in these results because many of them are quite nontrivial. For a more thorough discussion see [Voronkov 1998a, Voronkov 1999]. An analysis of first-order theorem proving based on rigid variables may be found in [Voronkov 1998b, Voronkov 1998a].

5.3. Other works

The first procedure for rigid E -unification described in [Gallier et al. 1988] has not been intended as an efficient procedure for rigid E -unification, but rather as a proof of its NP-completeness. Later, several papers described other algorithms for rigid E -unification [Goubault 1993, Beckert 1994, Becher and Petermann 1994, De Kogel 1995]. These algorithms have been presented as either more efficient or more simple than the original algorithm of Gallier et.al. The interest in papers on (non-simultaneous) rigid E -unification has been motivated by two reasons.

First, such algorithms were used in implementations of tableau provers with equality [e.g. Beckert 1994, Petermann 1994, Beckert 1997b]. Unlike Gallier et al.'s [1988] procedure, procedures of these papers use infinite sets of solutions or are non-terminating.

Second, for a long time there have been a hope to construct a decision procedure for simultaneous rigid E -unification by using finite complete sets of minimal solutions to rigid equations, maybe by changing the notion of a minimal solution. The undecidability result for simultaneous rigid E -unification has shown that this is not possible.

Moreover, it is hard to expect that there are decidable fragments of simultaneous rigid E -unification with a reasonably low complexity bound, except for the function-free fragment. Hence, such decision procedures are hardly interesting for theorem proving in classical logic. However, such procedures and a semi-decision procedure for simultaneous rigid E -unification are of a definite interest in connection with theorem proving in nonclassical predicate logics with equality, as explained in Section 9.

There are papers which define and study so-called *mixed E -unification* where some variables are treated as rigid and some as universal [Beckert 1994, Beckert 1997a, Beckert 1997b]. It is proposed to apply it in the situation when there are “formulas universal on a branch with respect to a variable”.

Two main sources of references to results on rigid E -unification are [Veanes 1997a, Voronkov 1999].

6. Incomplete procedures for rigid E -unification

In this chapter we describe an incomplete procedure for simultaneous rigid E -unification which is complete for theorem proving in first-order logic. This procedure was originally introduced by [Degtyarev and Voronkov 1996g], and improved in [Degtyarev and Voronkov 1997, Degtyarev and Voronkov 1998]. *In this section all formulas are assumed to be skolemized, i.e., contain no positive occurrences of \forall or negative occurrences of \exists .* We also assume that \approx is the only predicate symbol.

6.1. Gallier et.al.'s procedure: merits and problems

[Gallier et al. 1988, Gallier et al. 1990, Gallier et al. 1992, Gallier 1992] have shown how to use rigid E -unification in conjunction with equational matings. A corresponding formal description of a procedure is given, for example in [Gallier et al. 1992]. This procedure is based on the following premises:

1. For every rigid equation there exists a finite *complete set of minimal solutions*. This set is finite and can be computed by some terminating algorithm.
2. For every solvable system S of rigid equations, some of its solutions can be found incrementally (rigid-equation-wise) by a consecutive combination of finite complete sets of solutions for components of S .

Unfortunately, the second premise happened to be false, as noted by Becher and Petermann [1994]. To illustrate Gallier et al.'s [1988] procedure, we introduce one definition. Let Γ be a tableau branch.

- The *set of rigid equations on Γ* is defined in the following way. A rigid equation $E \vdash_{\forall} s \approx t$ is *on Γ* if $E \subseteq \{s' \approx t' \mid (T s' \approx t') \in \Gamma\}$ and $(F s \approx t) \in \Gamma$. (This definition is equivalent to the one given by Beckert and Hähnle [1992].)

Consider again Example 3.3. Gallier et.al.'s procedure would select one of the branches, for example the left one, and one rigid equation on that branch. In our case, there are two such rigid equations: $\vdash_{\forall} g(x, u, v) \approx g(y, fc, fd)$ and

$a \approx b \vdash_{\forall} g(x, u, v) \approx g(y, fc, fd)$. These rigid equations have the same complete set of minimal solutions consisting of one substitution $[x \mapsto y, u \mapsto fc, v \mapsto fd]$. According to the procedure, any substitution of the finite complete set of solutions is applied to the whole tableau, and the procedure is continued with the rest of the branches. It was asserted in the above mentioned papers that such a procedure gives a decision procedure for simultaneous rigid E -unification. In other terms, if a tableau is substitutively inconsistent, the procedure can make all branches closed without any applications of tableau expansion rules. But in our case, after the application of the substitution $[x \mapsto y, u \mapsto fc, v \mapsto fd]$ to the tableau we obtain the tableau which is not substitutively inconsistent any more. Thus, it cannot now be closed without expansions. It means that the second premise of Gallier et.al.' procedure is incorrect.

To close the tableau of this example, we need to consider a nonminimal solution to the second equation $a \approx b \vdash_{\forall} g(x, u, v) \approx g(y, fc, fd)$. If we choose the solution $[x \mapsto fa, y \mapsto fb, u \mapsto fc, v \mapsto fd]$ instead of a minimal one $[x \mapsto y, u \mapsto fc, v \mapsto fd]$, we would obtain a closed tableau. However, when we do not restrict ourselves by a complete set of minimal solutions, we lose termination.

We can note that *any* procedure that terminates for a given tableau cannot check whether the tableau is substitutively inconsistent, due to the undecidability of simultaneous rigid E -unification.

All these considerations do not imply that the procedure is incomplete for first-order logic with equality. Indeed, it is possible that after some additional applications of tableau expansion rules the procedure will find a solution. Using the language of matings, the solution can probably be found for some (not necessarily minimal) amplification.

Thus, Gallier et.al.'s procedure is an example of a tableau-based procedure for establishing provability in first-order logic with equality, which uses an incomplete but terminating algorithm for simultaneous rigid E -unification. This procedure generalizes tableau-based algorithms for logic with equality. Its main advantage is that substitutive inconsistency for the whole tableau can be found by an incremental branch closure, where branches are closed one by one. However, it is still unknown whether the Gallier et.al.'s procedure is complete for first-order logic with equality.

The use in tableau theorem proving of an incomplete procedure for (simultaneous) rigid E -unification in tableau-based methods was discussed in the literature [see e.g. Petermann 1994, Beckert 1995], especially after the first proof of the undecidability of simultaneous rigid E -unification [Degtyarev and Voronkov 1995e] that appeared in May 1995. It is still unknown if Gallier et.al.'s procedure is complete.

In the next section we describe a calculus that gives such a procedure.

6.2. An incomplete calculus for rigid E -unification

We will use the term "rigid basic superposition" to denote a "rigid" version of basic superposition. We formalize rigid basic superposition using constraints that is close to the presentation of Nieuwenhuis and Rubio [1995]. We will use ordering

constraints as defined in Section 2.2.

Below we will introduce an inference system **BSE** (Basic Superposition with Equality Solution) for solving rigid equations. The provable objects of **BSE** are *constrained rigid equations*. In this section we display constraints as sets of atomic equality and ordering constraints, instead of a conjunction.

- A *constrained rigid equation* is a pair consisting of a rigid equation R and a constraint C . Such a constrained rigid equation will be denoted $R \cdot C$.
- The calculus **BSE** consists of the following inference rules:
 - *Rigid basic (right and left) superposition rules* are the following inference rules:

$$\frac{E \cup \{s \approx t\} \vdash_{\forall} u[s'] \approx v \cdot C}{E \cup \{s \approx t\} \vdash_{\forall} u[t] \approx v \cdot C \cup \{s \succ t, u[s'] \succ v, s = s'\}} \quad (rbrs)$$

and

$$\frac{E \cup \{s \approx t, u[s'] \approx v\} \vdash_{\forall} e \cdot C}{E \cup \{s \approx t, u[t] \approx v\} \vdash_{\forall} e \cdot C \cup \{s \succ t, u[s'] \succ v, s = s'\}} \quad (rbls),$$

respectively.

- *Rigid equality solution* is the following inference rule:

$$\frac{E \vdash_{\forall} s \approx t \cdot C}{E \vdash_{\forall} s \approx s \cdot C \cup \{s = t\}} \quad (reqs)$$

Application of all the rules is restricted to the following conditions:

1. The constraint at the conclusion of the rule is satisfiable;
2. In $(reqs)$, $s \neq t$;
3. In the rigid basic superposition rules, the term s' is not a variable;
4. In $(rbls)$, $u[t] \neq v$.

- The *calculus BSE* is the logical calculus whose inference rules are $(rbls)$, $(rbrs)$ and $(reqs)$.

The basic restriction in **BSE** is formalized by representing most general unifiers through equality constraints. Condition 1 has two purposes. The satisfiability of equations in constraints is needed to preserve correctness of the method. The satisfiability of ordering constraints is needed to ensure termination (Theorem 6.3 below). Condition 2 is needed to forbid infinite chains of repeated applications of $(reqs)$. Conditions 3 and 4 are not essential, they are added as a standard optimization used in paramodulation-based methods.

- We denote by $R \cdot C \rightsquigarrow R' \cdot C'$ the fact that $R' \cdot C'$ is obtained from $R \cdot C$ by an application of an inference rule of **BSE**. The symbol \rightsquigarrow^* denotes the reflexive and transitive closure of \rightsquigarrow .

6.1. EXAMPLE. Consider the rigid equation $hx \approx a, ha \approx a, hb \approx fy \vdash_{\forall} y \approx gfy$. The ordering \succ is based on the precedence relation $f \succ_{\mathcal{F}} h \succ_{\mathcal{F}} b \succ_{\mathcal{F}} a$. Under this

ordering we have $hb \succ a$ and $fy \succ hb$. The following is a **BSE**-derivation for this rigid equation:

$$\begin{array}{c}
 \frac{hx \approx a, ha \approx a, hb \approx fy \vdash_{\vee} y \approx gfy \cdot \emptyset}{hx \approx a, ha \approx a, hb \approx fy \vdash_{\vee} y \approx ghb \cdot \{fy \succ hb, gfy \succ y, fy = fy\}} \quad (rbrs) \\
 \frac{hx \approx a, ha \approx a, hb \approx fy \vdash_{\vee} y \approx ga \cdot \{fy \succ hb, gfy \succ y, fy = fy, hx \succ a, ghb \succ y, hx = hb\}}{hx \approx a, ha \approx a, hb \approx fy \vdash_{\vee} y \approx y \cdot \{fy \succ hb, gfy \succ y, fy = fy, hx \succ a, ghb \succ y, hx = hb, y = ga\}} \quad (reqs)
 \end{array}$$

By using *constraint simplification*, i.e. replacement of constraints by equivalent “simpler” constraints we can rewrite this derivation as

$$\begin{array}{c}
 \frac{hx \approx a, ha \approx a, hb \approx fy \vdash_{\vee} y \approx gfy \cdot \emptyset}{hx \approx a, ha \approx a, hb \approx fy \vdash_{\vee} y \approx ghb \cdot \emptyset} \quad (rbrs) \\
 \frac{hx \approx a, ha \approx a, hb \approx fy \vdash_{\vee} y \approx ga \cdot \{ghb \succ y, x = b\}}{hx \approx a, ha \approx a, hb \approx fy \vdash_{\vee} y \approx y \cdot \{x = b, y = ga\}} \quad (reqs)
 \end{array}$$

Soundness of **BSE** can be formulated in the following way:

6.2. THEOREM (Soundness of BSE). *Let $R \cdot \emptyset \rightsquigarrow^* E \vdash_{\vee} s \approx s \cdot C$. Then every substitution satisfying C is a solution to R . In particular, R is solvable.*

This theorem leads to the following definition.

► A constraint C is called an *answer constraint* for a rigid equation R if for some rigid equation $E \vdash_{\vee} s \approx s$ we have $R \cdot \emptyset \rightsquigarrow^* E \vdash_{\vee} s \approx s \cdot C$.

We note that **BSE** is an incomplete calculus for solving rigid equations. It means that there are solvable rigid equations R that have no answer constraint. For instance, consider the rigid equation⁵ $x \approx a \vdash_{\vee} gx \approx x$. It has one solution $[x \mapsto ga]$. However, no inference rule of **BSE** is applicable to this rigid equation.

Although the calculus **BSE** is incomplete, it has some pleasant termination properties:

6.3. THEOREM (Termination of BSE). *For any constrained rigid equation $R \cdot C$, there exists a finite number of derivations from $R \cdot C$.*

This property is due to the constraints: after some finite number of steps we will not be able to apply any inference rule because the constraint in the conclusion of any rule would be unsatisfiable. Ordering constraints are not needed for either soundness or completeness of **BSE**. The pragmatics behind ordering constraints is to ensure that the search for solutions to a rigid equation is finite. In addition, the use of ordering constraints prunes the search space.

⁵Suggested by G.Bechev (private communication).

In [Degtyarev and Voronkov 1996g] left rigid basic superposition was formulated incorrectly in the following way:

$$\frac{E \cup \{s \approx t, u[s'] \approx v\} \vdash_{\forall} e \cdot C}{E \cup \{s \approx t, u[s'] \approx v, u[t] \approx v\} \vdash_{\forall} e \cdot C \cup \{s \succ t, u[s'] \succ v, s = s'\}} \text{ (rbls)}$$

With this formulation, termination is not guaranteed as shown in [Degtyarev and Voronkov 1997, Degtyarev and Voronkov 1998].

To illustrate this theorem, consider again Example 6.1. The rigid equation of this example has an infinite number of solutions including $[x \mapsto a, y \mapsto gh^n a]$, for every natural number n . However, all possible BSE-derivations starting with $hx \approx a, ha \approx a, hb \approx fy \vdash_{\forall} y \approx gfy \cdot \emptyset$ give only two answer constraints, one is

$$\{fy \succ hb, gfy \succ y, fy = fy, hx \succ a, ghb \succ y, hx = hb, y = ga\}$$

shown in Example 6.1, another is $\{fy \succ hb, gfy \succ y, fy = fy, y = ghb\}$ obtained from the following derivation:

$$\frac{\frac{hx \approx a, ha \approx a, hb \approx fy \vdash_{\forall} y \approx gfy \cdot \emptyset}{hx \approx a, ha \approx a, hb \approx fy \vdash_{\forall} y \approx ghb \cdot \{fy \succ hb, gfy \succ y, fy = fy\}} \text{ (rbrs)}}{hx \approx a, ha \approx a, hb \approx fy \vdash_{\forall} y \approx y \cdot \{fy \succ hb, gfy \succ y, fy = fy, y = ghb\}} \text{ (reqs)}$$

This answer constraint can be simplified to $\{y = ghb\}$. There are some other derivations with the same answer constraints, but only a finite number.

Theorem 6.3 yields

6.4. THEOREM. *Any rigid equation has a finite number of answer constraints. There is an algorithm giving by any rigid equation R the set of all answer constraints for R .*

6.3. Answer constraints and the tableau method

In this section we consider how to use the system BSE for theorem proving by the tableau method. Since we only consider skolemized formulas, we have no δ -rules in tableau calculi.

We extend the notion of answer constraints to tableau branches. Let Γ be such a branch.

- A constraint C is a *answer constraint* for Γ if C is an answer constraint for some rigid equation on Γ .

By Theorem 6.4, we obtain

6.5. THEOREM. *Any tableau branch has a finite number of answer constraints. There is an algorithm giving by any tableau branch Γ the set of all answer constraints for Γ .*

Note that every substitution satisfying an answer constraint for Γ is a substitution closing Γ .

The following theorem asserts soundness and completeness of the tableau method with answer constraints [Degtyarev and Voronkov 1997, Degtyarev and Voronkov 1998]:

6.6. THEOREM. *Let φ be a sentence. Then φ is provable in first-order logic with equality if and only if there is a tableau \mathcal{T} obtained from $F\varphi$ by tableau expansion rules with the following property. Let $\Gamma_1, \dots, \Gamma_n$ be all branches of \mathcal{T} . Then there exist answer constraints C_1, \dots, C_n for $\Gamma_1, \dots, \Gamma_n$, respectively, such that $C_1 \cup \dots \cup C_n$ is satisfiable.*

Note that if the constraint $C_1 \cup \dots \cup C_n$ is satisfiable, then there exists a substitution θ satisfying this constraint. Since θ satisfies each C_i , it closes all branches of the tableau.

6.7. EXAMPLE. To illustrate this theorem, consider again the tableau of Example 3.3. We will use the lexicographic path ordering \succ based on the precedence $g \succ_{\mathcal{F}} f \succ_{\mathcal{F}} a \succ_{\mathcal{F}} b \succ_{\mathcal{F}} c \succ_{\mathcal{F}} d$.

There is one rigid equation on each branch of the tableau:

$$a \approx b \vdash_{\forall} g(x, u, v) \approx g(y, fc, fd); \quad (6.1)$$

$$c \approx d \vdash_{\forall} g(u, x, y) \approx g(v, fa, fb). \quad (6.2)$$

Rigid basic superposition is applicable to none of these rigid equations. Rigid equation (6.1) has one answer constraint $\{g(x, u, v) = g(y, fc, fd)\}$ obtained by the following application of the rigid equality solution rule:

$$\frac{a \approx b \vdash_{\forall} g(x, u, v) \approx g(y, fc, fd) \cdot \emptyset}{a \approx b \vdash_{\forall} g(x, u, v) \approx g(x, u, v) \cdot \{g(x, u, v) = g(y, fc, fd)\}} \text{ (reqs)}$$

Similarly, rigid equation (6.2) has one answer constraint $\{g(u, x, y) = g(v, fa, fb)\}$. The union of these constraints $\{g(x, u, v) = g(y, fc, fd), g(u, x, y) = g(v, fa, fb)\}$ is unsatisfiable. Thus, our method find no solution for this substitutively inconsistent tableau. After more tableau expansion steps we obtain a tableau whose atomic part is shown in Figure 12.

It has four branches. The multisets of literals on the branches are

$$\begin{aligned} S_1 &= \{a \approx b, g(x_1, u_1, v_1) \not\approx g(y_1, fc, fd), g(x_2, u_2, v_2) \not\approx g(y_2, fc, fd)\}; \\ S_2 &= \{a \approx b, c \approx d, g(x_1, u_1, v_1) \not\approx g(y_1, fc, fd), g(u_2, x_2, y_2) \not\approx g(v_2, fa, fb)\}; \\ S_3 &= \{a \approx b, c \approx d, g(u_1, x_1, y_1) \not\approx g(v_1, fa, fb), g(x_3, u_3, v_3) \not\approx g(y_3, fc, fd)\}; \\ S_4 &= \{c \approx d, g(u_1, x_1, y_1) \not\approx g(v_1, fa, fb), g(u_3, x_3, y_3) \not\approx g(v_3, fa, fb)\}. \end{aligned}$$

Consider the following rigid equations R_1 – R_4 on the branches S_1 – S_4 , respectively:

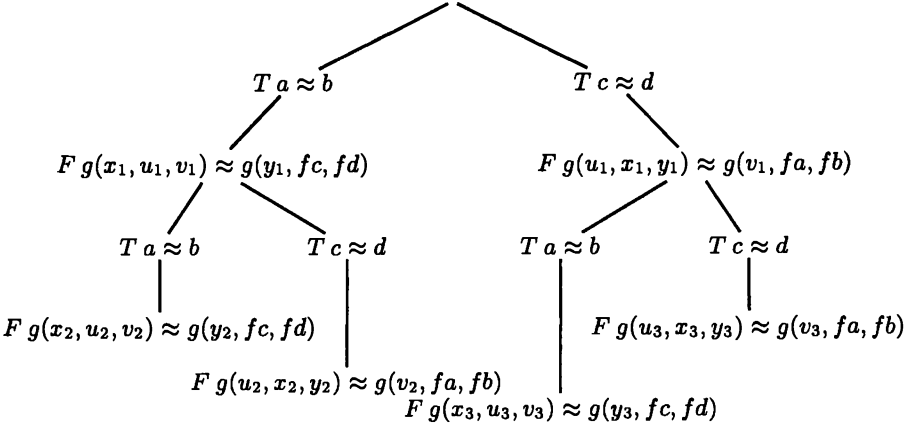


Figure 12: Atomic part of a tableau

$$\begin{aligned}
 R_1 &= a \approx b \vdash_{\forall} g(x_2, u_2, v_2) \approx g(y_2, fc, fd); \\
 R_2 &= a \approx b, c \approx d \vdash_{\forall} g(x_1, u_1, v_1) \approx g(y_1, fc, fd); \\
 R_3 &= a \approx b, c \approx d \vdash_{\forall} g(u_1, x_1, y_1) \approx g(v_1, fa, fb); \\
 R_4 &= c \approx d \vdash_{\forall} g(u_3, x_3, y_3) \approx g(v_3, fa, fb).
 \end{aligned}$$

Apply the following **BSE**-derivations to R_1 – R_4 :

$$\begin{array}{c}
 \frac{a \approx b \vdash_{\forall} g(x_2, u_2, v_2) \approx g(y_2, fc, fd) \cdot \emptyset}{a \approx b \vdash_{\forall} g(x_2, u_2, v_2) \approx g(x_2, u_2, v_2) \cdot \{g(x_2, u_2, v_2) = g(y_2, fc, fd)\}} \text{ (reqs)} \\
 \\
 \frac{\frac{a \approx b, c \approx d \vdash_{\forall} g(x_1, u_1, v_1) \approx g(y_1, fc, fd) \cdot \emptyset}{a \approx b, c \approx d \vdash_{\forall} g(x_1, u_1, v_1) \approx g(y_1, fd, fd)} \text{ (rbrs)}}{\cdot \{c \succ d, g(y_1, fc, fd) \succ g(x_1, u_1, v_1), c = c\}} \text{ (reqs)} \\
 \hline
 a \approx b, c \approx d \vdash_{\forall} g(x_1, u_1, v_1) \approx g(x_1, u_1, v_1) \\
 \cdot \{c \succ d, g(y_1, fc, fd) \succ g(x_1, u_1, v_1), c = c, g(x_1, u_1, v_1) = g(y_1, fd, fd)\} \\
 \\
 \frac{\frac{a \approx b, c \approx d \vdash_{\forall} g(u_1, x_1, y_1) \approx g(v_1, fa, fb) \cdot \emptyset}{a \approx b, c \approx d \vdash_{\forall} g(u_1, x_1, y_1) \approx g(v_1, fb, fb)} \text{ (rbrs)}}{\cdot \{a \succ b, g(v_1, fa, fb) \succ g(u_1, x_1, y_1), a = a\}} \text{ (reqs)} \\
 \hline
 a \approx b, c \approx d \vdash_{\forall} g(u_1, x_1, y_1) \approx g(u_1, x_1, y_1) \\
 \cdot \{a \succ b, g(v_1, fa, fb) \succ g(u_1, x_1, y_1), a = a, g(u_1, x_1, y_1) = g(v_1, fb, fb)\}
 \end{array}$$

$$\frac{c \approx d \vdash_{\forall} g(u_3, x_3, y_3) \approx g(v_3, fa, fb) \cdot \emptyset}{c \approx d \vdash_{\forall} g(u_3, x_3, y_3) \approx g(v_3, x_3, y_3) \cdot \{g(u_3, x_3, y_3) = g(v_3, fa, fb)\}} \text{ (reqs)}$$

The union of the answer constraints of these derivations is

$$\begin{aligned} & \{g(x_2, u_2, v_2) = g(y_2, fc, fd), \\ & c \succ d, g(y_1, fc, fd) \succ g(x_1, u_1, v_1), c = c, g(x_1, u_1, v_1) = g(y_1, fd, fd), \\ & a \succ b, a = a, g(v_1, fa, fb) \succ g(u_1, x_1, y_1), g(u_1, x_1, y_1) = g(v_1, fb, fb), \\ & g(u_3, x_3, y_3) = g(v_3, fa, fb)\}. \end{aligned}$$

It is easy to check that the following substitution solves all the constraints:

$$\begin{aligned} & [x_1 \mapsto fb, y_1 \mapsto fb, u_1 \mapsto fd, v_1 \mapsto fd, x_2 \mapsto b, y_2 \mapsto b, \\ & u_2 \mapsto fc, v_2 \mapsto fd, u_3 \mapsto d, v_3 \mapsto d, x_3 \mapsto fa, y_3 \mapsto fb]. \end{aligned}$$

Hence, this substitution closes all the branches of the tableau.

7. Sequent-based calculi and paramodulation

In order to construct a tableau-based free-variable system with equality, in Section 5 we used a way which can be characterized as *total theory reasoning* [Stickel 1985]. Indeed, branch closure (or closure of the whole tableau) has been represented as one derivation step. This step included the detection of a literal set which is inconsistent in the equality theory. In contrast with this approach, the ground version $\mathbf{LK}^=$ of tableaux with equality is, in the same terminology, an example of *partial theory reasoning*. In $\mathbf{LK}^=$, the search for inconsistency of a sequent in the equality theory is realized through a sequence of derivation steps, mainly the application of the rule (\approx) of replacement of equals by equals. A sequence of applications of this rule is intended to give a syntactically inconsistent sequent, i.e. a sequent inconsistent in the empty theory.

Within the resolution framework, paramodulation has been defined as a generalization of replacement of equals by equals. For free variable tableau-based systems such rules have first been formulated by Loveland [1978] in the context of model elimination and by Fitting [1990] in the context of tableaux by the name of the *mgu tableau replacement rule*. Characterizing this rule Fitting writes:

It was necessary to modify the tableau system with equality by introducing free variables in order to get a version suitable for implementation. We must do a similar thing with the resolution system. The result is a variation of what is known in the literature as *paramodulation* ...

7.1. Fitting's tableau paramodulation

Since paramodulation has appeared before and is a generally accepted term, we will call the mgu tableau replacement rule simply *tableau paramodulation*.

► *Tableau paramodulation* is the following inference rule:

$$\frac{\Gamma_1, X A[s'], T s \approx t \mid \dots \mid \Gamma_n}{(\Gamma_1, X A[s'], T s \approx t, X A[t] \mid \dots \mid \Gamma_n) \text{mgu}(s, s')} \text{ (tpar)}$$

where A is an atomic formula.

In the tableau paramodulation rule, the substitution $\text{mgu}(s, s')$ is applied to the whole tableau, i.e. it is “global”. Free variables in tableaux are treated *rigidly*, as placeholders for terms, or unknown parameters, but not as universally quantified variables.

Besides the tableau paramodulation rule, the system of Fitting for equational logic includes the following two inference rules.

► *Free variable tableau reflexivity* of Fitting [1990] is the following inference rule:

$$\frac{\Gamma_1 \mid \dots \mid \Gamma_n}{\Gamma_1, T x \approx x \mid \dots \mid \Gamma_n}$$

► *Tableau function reflexivity* of Fitting [1990] is the following inference rule:

$$\frac{\Gamma_1 \mid \dots \mid \Gamma_n}{\Gamma_1, T f(x_1, \dots, x_n) \approx f(x_1, \dots, x_n) \mid \dots \mid \Gamma_n}$$

It is easy to see that instead of the last two rules one can change the initial tableau for proving a formula ξ from $F \xi$ to

$$\{T \forall \bar{x} (f(\bar{x}) \approx f(\bar{x})) \mid f \in \mathcal{F}, T \forall x (x \approx x), F \xi\}$$

Below we will consider some difficulties in removing the function reflexivity rule from Fitting’s system.

► Fitting’s *free variable tableau system* $\mathbf{TK}^=$ for logic with equality consists of the rules of \mathbf{TK} plus the tableau paramodulation rule, free variable tableau reflexivity rule and the tableau function reflexivity rule.⁶

This system is complete:

7.1. THEOREM. *Let ξ be a closed formula. The following conditions are equivalent:*

1. ξ is provable in first-order logic with equality;
2. there is a derivation of the empty tableau from the tableau $F \xi$ in $\mathbf{TK}^=$.

The system $\mathbf{TK}^=$ has all disadvantages of the early paramodulation rule of Robinson and Wos [1969]:

1. Function reflexivity rule is used;

⁶An inessential difference between our formulation of Fitting’s system and Fitting’s [1990] formulation, as well as of [Fitting 1996] is that we remove closed branches from the tableau while in the system of [Fitting 1990] one can close the same branch several times. Also, Fitting does not give a name to his system, we call it here $\mathbf{TK}^=$ for convenience.

2. Paramodulation into variables is allowed;
3. Increasing applications of paramodulation are allowed (for example, x can be rewritten to fx using an equation $x \approx fx$).

As a consequence, for a given tableau expansion there may be an infinite sequence of paramodulations, either due to the use of function reflexivity or due to the use of increasing applications of paramodulation.

Fitting uses partial theory reasoning for adding equality to free-variable tableaux. He extended the tableau calculus \mathbf{TK} by simple rules for equality. However, his scheme of proof-search is, in fact, based on finding all solutions for simultaneous rigid E -unification. Fitting's completeness theorem is formulated in the way similar to Theorem 3.1 and leads to a proof-search strategy consisting of two parts. In the first part a tableau is constructed using tableau expansion rules. In the second part, equality rules and branch closure rules are applied with the purpose of closing the tableau. The system $\mathbf{TK}^=$ has the following very strong property (a reformulation of Lifting Lemma of Fitting [1990], page 220).

7.2. THEOREM. *If a tableau \mathcal{T} is substitutively inconsistent, then there is derivation of the empty tableau from \mathcal{T} in $\mathbf{TK}^=$ not using tableau expansion rules.*

Unlike \mathbf{TK} , the system $\mathbf{TK}^=$ may have *infinite derivations without tableau expansion rules*, i.e. the proof-search for a given tableau expansion may be nonterminating. It is not possible to restrict paramodulation in $\mathbf{TK}^=$ to obtain a system S which satisfies Theorem 7.2 and has the termination property for the subsystem of S without tableau expansion rules: any such system can be used as a decision procedure for simultaneous rigid E -unification. Moreover, if we drop function reflexivity, the resulting system would not satisfy Theorem 7.2:

7.3. EXAMPLE. Consider the tableau of Example 3.3. Its atomic part has the form

$$T a \approx b, F g(x, u, v) \approx g(y, fc, fd) \mid T c \approx d, F g(u, x, y) \approx g(v, fa, fb).$$

This tableau is substitutively inconsistent. However, we cannot derive the empty tableau from it without the use of tableau function reflexivity.

Plaisted [1995] described a tableau system which is complete, does not use function reflexivity and has the termination property for the subsystem without tableau expansion rules but uses a new tableau factoring rule. [Degtyarev and Voronkov 1997, Degtyarev and Voronkov 1998] described a complete system obtained from the $\mathbf{TK}^=$ by dropping tableau function reflexivity and imposing several restrictions on paramodulation. The restrictions ensure the termination property. In particular, results of [Degtyarev and Voronkov 1997, Degtyarev and Voronkov 1998] imply that $\mathbf{TK}^=$ is complete without function reflexivity. These results are described below.

An attempt to improve Fitting's tableau paramodulation was made in [Beckert and Hähnle 1992]. In that paper a restricted form of paramodulation is introduced

in which “it is not necessary to apply equalities to other equalities” and function reflexivity is not used. This method is nonterminating for a given tableau expansion. In addition, despite the claim of completeness, the method is incomplete. For example, as noted in [Degtyarev and Voronkov 1996g, Degtyarev and Voronkov 1998], the method cannot prove the formula $\exists x(a \approx b \wedge g(fa, fb) \approx h(fa, fb) \supset g(x, x) \approx h(x, x))$.

7.2. Tableau basic superposition

In this section we describe a calculus introduced in [Degtyarev and Voronkov 1996g, Degtyarev and Voronkov 1997, Degtyarev and Voronkov 1998]. This calculus imposes the following restrictions on paramodulation:

1. no function reflexivity is needed;
2. paramodulation into variables is not allowed;
3. orderings are used so that there are no increasing applications of paramodulation;
4. basic restriction on paramodulation allows one to prohibit paramodulation into nonvariable terms introduced by unification.

We will introduce the logical calculus **TBSE** (Tableau Basic Superposition with Equality Solution) which is an adaptation of the calculus **BSE** of Section 6.2 to tableaux. The provable objects of **BSE** are *constrained tableaux*.

- A *constrained tableau* is a pair consisting of a tableau \mathcal{T} and a constraint \mathcal{C} . Such a constrained tableau will be denoted $\mathcal{T} \cdot \mathcal{C}$.

In this section we display constraints as sets of atomic equality and ordering constraints, instead of a conjunction.

In this section we assume that all rules of **TK** are modified for constraint tableaux such that the tableau expansion rules do not change the constraint. For example, one of the β -rules is changed to

$$\frac{\Gamma_1, T \varphi \vee \psi \mid \dots \mid \Gamma_n \cdot \mathcal{C}}{\Gamma_1, T \varphi \mid \Gamma_1, T \psi \mid \dots \mid \Gamma_n \cdot \mathcal{C}} (\beta),$$

and the rule (abc) is changed to

$$\frac{\Gamma_1, T A, F B \mid \Gamma_2 \mid \dots \mid \Gamma_n \cdot \mathcal{C}}{\Gamma_2 \mid \dots \mid \Gamma_n \cdot \mathcal{C} \cup \{A = B\}} (abc),$$

The calculus **TBSE** consists of the following inference rules.

- *Tableau basic superposition* is the following inference rule:

$$\frac{\Gamma_1, T s \approx t, X u[s'] \approx v \mid \dots \mid \Gamma_n \cdot \mathcal{C}}{\Gamma_1, T s \approx t, X u[t] \approx v \mid \dots \mid \Gamma_n \cdot \mathcal{C} \cup \{s \succ t, u[s'] \succ v, s = s'\}} (tbs),$$

where

1. The constraint at the conclusion of the rule is satisfiable;
2. The signed formula $X u[t] \approx v$ does not occur in Γ_1 ;
3. The term s' is not a variable.

► *Tableau equality solution* is the following inference rule:

$$\frac{\Gamma_1, F s \approx t \mid \dots \mid \Gamma_n \cdot C}{\Gamma_2 \mid \dots \mid \Gamma_n \cdot C \cup \{s = t\}} \text{ (teqs)}$$

where $C \cup \{s = t\}$ is satisfiable.

► The *calculus TBSE* is the logical calculus whose inference rules are *(tbs)*, *(teqs)* and all rules of **TK**.

7.4. THEOREM (Soundness and completeness of **TBSE**). *Let ξ be a formula. It is provable in first-order logic with equality if and only if there is a derivation from the constrained tableau $F \xi \cdot \emptyset$ of a constraint tableau of the form $\# \cdot C$.*

This logical system has a pleasant termination property:

7.5. THEOREM (Termination of **TBSE**). *For every constrained tableau $\mathcal{T} \cdot C$ there is only a finite number of derivations from $\mathcal{T} \cdot C$ not using the tableau expansion rules.*

This means that for a given amplification we cannot have infinite search. Infinite search without expansion steps is possible in Fitting's system.

To illustrate the connection between the tableau basic superposition rules and rules of **TBSE**, we reconsider the tableau of Example 3.3. On the branch containing the literal $g(x_1, u_1, v_1) \not\approx g(y_1, fc, fd)$ and the equation $c \approx d$, we can apply rigid basic superposition that adds $g(x_1, u_1, v_1) \not\approx g(y_1, fd, fd)$ to the branch. Similarly, we can apply rigid basic superposition to the branch containing $g(u_1, x_1, y_1) \not\approx g(v_1, fa, fb)$ and $a \approx b$, obtaining $g(u_1, x_1, y_1) \not\approx g(v_1, fb, fb)$. This results in the tableau displayed in the tree-like form (we only show the atomic formulas in the tableau) shown in Figure 13

After four application of the tableau equality solution rules all branches of this tableau become closed. The resulting constraint of this derivation is the same as the union of the answer constraints of Example 6.7 on page 658.

7.3. Rigid variables in resolution theorem proving

In Section 3.1 we have defined the notion of substitutive inconsistency for branches and tableaux. Originally, this notion has been introduced by Chang [1972] and Lee and Chang [1973] for sets of clauses.

► A finite set or a multiset of clauses S is called *substitutively inconsistent* if there is a substitution σ such that the formula $\bigwedge_{C \in S} C\sigma$ is inconsistent.

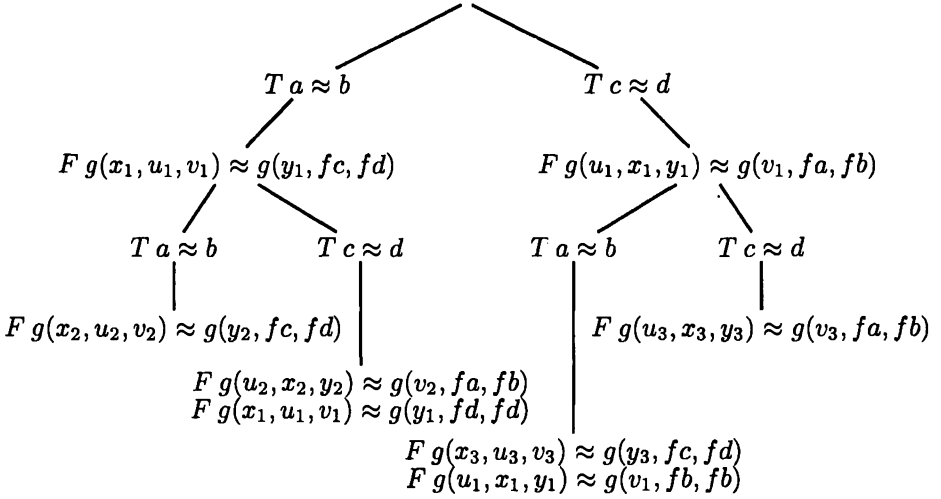


Figure 13: Atomic part of a tableau

Later, Plaisted [1995] called substitutive inconsistency rigid Eq-unsatisfiability. For establishing substitutive inconsistency, Chang [1972] used so-called *V*-resolution, and Lee and Chang [1973] *V*-resolution with *V*-paramodulation, where *V* stands for “variable-constrained”. Following Becher and Petermann [1994], instead of “variable-constrained” we will use more accepted words “rigid resolution” and “rigid paramodulation” for *V*-resolution and *V*-paramodulation.

Rigid resolution (respectively, paramodulation) is a “rigid” variant of resolution (respectively, paramodulation). In these inference rules, all variables are treated “rigidly”, as unknown parameters while in the ordinary resolution and paramodulation rules variables are treated as universally quantified. Hence, similar to tableaux, we will formulate these inference rules as working on multisets of clauses.

► The *calculus RRP* consists of the following four rules:

► *rigid resolution rule*:

$$\frac{S \cup \{A \vee C_1\} \cup \{\neg A' \vee C_2\}}{(S \cup \{A \vee C_1\} \cup \{\neg A' \vee C_2, C_1 \vee C_2\})\text{mgu}(A, A')} \text{ (rres);}$$

► *rigid factoring rule*:

$$\frac{S, \cup\{L \vee L' \vee C\}}{(S \cup \{L \vee C\})\text{mgu}(L, L')} \text{ (rfac);}$$

► *rigid reflexivity rule*:

$$\frac{S \cup \{s \not\approx t \vee C\}}{(S \cup \{C\})\text{mgu}(s, t)} \text{ (rrefl);}$$

► *rigid paramodulation rule:*

$$\frac{S \cup \{L[s'] \vee C_1\} \cup \{s \approx t \vee C_2\}}{(S \cup \{L[s'] \vee C_1\} \cup \{s \approx t \vee C_2\} \cup \{L[t] \vee C_1 \vee C_2\})\text{mgu}(s, s')} \text{ (rpar)}.$$

The goal is to derive a multiset of clauses containing the empty clause \square . The calculus **RRP** is complete for checking substitutive inconsistency in logic without equality.⁷ For logic with equality, completeness of **RRP** for checking substitutive inconsistency can be proved with the additional function reflexivity rule, similar to the (*tfr*) rule for tableaux:

$$\frac{S}{S \cup \{f(x_1, \dots, x_n) \approx f(x_1, \dots, x_n)\}} \text{ (rfr)},$$

where f is a function symbol occurring in S of arity $n > 0$ and x_1, \dots, x_n are new variables. Instead of this rule we can use

$$\frac{S}{S[x \mapsto f(x_1, \dots, x_n)]}.$$

We have

7.6. THEOREM. *Let S be a finite multiset of clauses. Then the following conditions are equivalent:*

1. S is substitutively inconsistent;
2. *there is a multiset of clauses S' such that S' is derivable from S in **RRP** + (rfr) and $\square \in S'$.*

7.7. EXAMPLE. Following [Degtyarev and Voronkov 1998] we demonstrate that **RRP** is incomplete for checking substitutive inconsistency. This example is a translation of Example 7.3 on page 662 to clausal form. In fact, it encodes substitutive inconsistency of the tableau from that example via a multiset of clauses.

$$\begin{aligned} a &\approx b \vee c \approx d \\ a &\approx b \vee g(u, x, y) \not\approx g(v, fa, fb) \\ c &\approx d \vee g(x, u, v) \not\approx g(y, fc, fd) \\ g(x, u, v) &\not\approx g(y, fc, fd) \vee g(u, x, y) \not\approx g(v, fa, fb) \end{aligned}$$

This multiset of clauses is substitutively inconsistent. The only substitution making this multiset inconsistent is $\theta \approx [x \mapsto fa, y \mapsto fb, u \mapsto fc, v \mapsto fd]$. The reader can check that any application of an inference rule of **RRP** to this multiset of clauses gives a substitution incompatible with θ .

In the calculus **RRP** the rigid paramodulation rule is unrestricted: there are no ordering restrictions and paramodulation into variables is allowed⁸.

⁷Lee and Chang [1973] assert without proof a theorem which implies completeness of **RRP** for logic with equality which is not true as we show later.

⁸Plaisted [1995] shows incompleteness of ordered rigid paramodulation and notes that this incompleteness result may be not true for unrestricted rigid paramodulation. Thus, our example improves the result of [Plaisted 1995].

7.4. From resolution to tableaux and matings

Several papers compared resolution with connections (matings), for example [Bibel 1981, Eder 1988, Eder 1991, Mints 1990, Baumgartner and Furbach 1993, Avron 1993]. In particular, Eder [1988] has shown “how a resolution refutation can be transformed to a complementary connected matrix, thus yielding a connection derivation of a given matrix”. Under this translation, the connection calculus had to be augmented with a factoring (factorization) rule: “In cases where resolution steps are combined with factorization steps we have to introduce factorization links in addition to the connections”.

Plaisted [1995] uses a similar technique to translate “rigid resolution and paramodulation” into the connection proofs using the following idea. Let S be a multiset of clauses. Plaisted calls an *amplification* of S a multiset T of one or more variants of clauses in S with variables renamed so that each variable appears in at most one clause in T . (The same notion has been used in V -resolution of Lee and Chang [1973] by the name of “alleged substitutively unsatisfiable set”.) For large enough T , substitutive inconsistency of T is equivalent to the inconsistency of S . Then the obtained rigid refutation is translated into so-called *path paramodulation* over the multiset of paths through T . This technique is, in fact, identical to the technique used by Eder [1991]. For example, Plaisted’s “path correspondence” was used by Eder as “shortening of a path”. Final results of Plaisted are also similar to the results of Eder: the tableau system of Plaisted contains in addition to the standard tableau rules a factoring rule.

As a consequence, tableau calculi with factoring do not allow one to close the tableau “branch-wise” as it is done in the procedure of Gallier et.al. or in the calculus **BSE** of Section 6. In other words, path paramodulation of Plaisted does not allow branch-wise computation of solutions to rigid E -unification for separate branches. Nevertheless, this technique gives some substitute for tableau basic superposition described here, because path paramodulation terminates for a given tableau expansion. Unlike tableau basic superposition, the termination of path paramodulation is due to the fact that both literals involved in paramodulation are deleted from their path.

8. Equality elimination

In all sequent-based free-variable methods, proof-search comprises two kinds of operations. Operations (or inferences rules) of the first kind construct a tree of sequents, matrix or tableau using suitable *expansion rules*. The second kind of operations (not always formalized as inference rules) tries to close leaf sequents in the tree, paths in the matrix or branches in the tableau using suitable substitutions.

Despite differences among these methods, they have a common disadvantage: “globality” of the second kind of operations. Maslov et al. [1983] characterized such methods (tableau, matings and model elimination) as *global*, and methods like resolution or the inverse method as *local*. We quote Maslov et al. [1983]:

... Following the idea it is not the proof we get at first but some intermediate object — prededuction; after that we verify the possibility of turning the sufficiently complicated predeductions into correct deductions (and this amount of work is generally speaking useless for the further steps if the given prededuction cannot still be turned into a correct deduction and have to be built over). This is connected first of all with the “globality” of work with the predeductions. That is why more perspective are apparently the methods enabling us to combine the unpreciseness of the values of variables and “local independence” of work (i.e. the using of a relatively small part of deduction at the given stage of work and independence of this work from the remaining parts of the deduction).

Consider several examples. In the MGU replacement rule of Fitting [1990] the substitution produced by unification is applied to the whole tableau. In equational matings [Gallier et al. 1989, Gallier et al. 1990] the substitution is global for all paths in a matrix. In tableau basic superposition of Section 7.2 the constraint is global for the whole tableau and every application of rigid basic superposition on any branch changes the constraint and thus influences all other branches.

In this chapter we describe the *equality elimination method* which allows us to localize equality reasoning so that we obtain a partially local sequent-based method (for tableaux or matings) or a completely local sequent-based method (for the inverse method). This method is based on extending sequent-based provers by a bottom-up equation solver using basic superposition. Solutions to equations are generated by this solver and used to close branches of a tableau, paths of a matrix or to generate axioms for the inverse method. The equation solution is even more restricted by the use of orderings, basic simplification and subsumption.

The equality elimination method was originally introduced in [Degtyarev and Voronkov 1994b, Degtyarev and Voronkov 1995c] as a method of handling equality in logic programs. Later, it has been applied to the inverse method in [Degtyarev and Voronkov 1995a] and to matings and semantic tableaux in [Degtyarev and Voronkov 1995b, Degtyarev and Voronkov 1996b]. Applications of equality elimination to the tableau method and to the inverse method are based on a common characterization of provability in terms of solution clauses described in various forms in the rest of this section. This characterization can be considered as a computationally improved version of the Herbrand theorem. However, the resulting calculi for the inverse method and for the tableau method are quite different because the inverse method is local while the tableau method is global.

8.1. Theoretical basis for equality elimination

For the rest of this section we assume that ξ denotes a closed formula in the skolem negation normal form to be proved (the “goal”). We assume that all different occurrences of quantifiers in ξ bind different variables. For example, ξ cannot have the form $\exists xA \vee \exists xB$. We will identify subformulas of ξ and their superformulas with their *occurrences* in ξ . For example, in the formula ξ of the form $A \wedge (A \vee B)$ the second occurrence of A is considered a subformula of the occurrence of $(A \vee B)$, but the first occurrence of A is not.

In order to define a more efficient sequent-based calculus, we will only deal with some subformulas of ξ called conjunctive subformulas.

- An (occurrence of a) subformula φ of ξ is called *conjunctive* if it is an occurrence in a subformula $\varphi \wedge \psi$ or in $\psi \wedge \varphi$.

In fact, conjunctive subformulas of ξ are equivalent to β -subformulas.

- We call a formula φ a *superformula* of a formula ψ if ψ is a subformula of φ .
- Let φ be a subformula of ξ . A *conjunctive superformula* of φ in ξ is any superformula ψ of φ that is a conjunctive subformula of ξ .
- The *least conjunctive superformula* of φ in ξ is the conjunctive superformula ψ of φ in ξ such that any other conjunctive superformula of φ is a superformula of ψ .

Note that any formula having a conjunctive superformula has the unique least conjunctive superformula. We also note that if φ is a formula and ψ is its least conjunctive superformula, then $\varphi \vdash \psi$. This partially explains the need for introducing least conjunctive superformulas. There are deterministic chains of inferences in sequent systems consisting of α - and γ -rules. For example,

$$\frac{\frac{\Gamma, F \varphi}{\Gamma, F \varphi \vee \psi} (\alpha)}{\Gamma, F \exists x(\varphi \vee \psi)} (\gamma)$$

By restricting ourselves to conjunctive superformulas only, we eliminate these deterministic chains, making them in one step⁹

Since we fix the goal formula ξ , we will speak about the least conjunctive superformula of φ meaning the least conjunctive superformula of φ in ξ . Least conjunctive superformulas are illustrated in Table 3 (ignore for a moment the third column of the table).

We can enumerate all conjunctive subformulas ξ_1, \dots, ξ_n of ξ , for example in the order of their occurrences in ξ . Thus we can unambiguously use “the k th conjunctive (sub)formula” ξ_k of ξ .

Let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be predicate symbols not occurring in ξ .

- We say that the atomic formula $\mathcal{A}_k(x_1, \dots, x_m)$ is the ξ -*name* of a subformula φ of ξ if

1. the least conjunctive superformula of φ is ξ_k ;

⁹This simple but powerful idea of restricting to conjunctive superformulas [Voronkov 1992] has been described in the form of a sequent calculus in [Voronkov 1985, Voronkov 1990b] and implemented in the theorem prover described in [Voronkov 1990a], in a theorem prover for intuitionistic logic by Tammet [1996], and a theorem prover for modal logic K [Voronkov 2000a, Voronkov 2001a]. In the framework of tableau theorem proving, several papers used permutabilities of inference rules in sequent calculi [see e.g. Shankar 1992]. In fact, the least conjunctive superformula of φ is a superformula ψ of φ provable from φ and such that all inference rules applied in the proof of ψ from φ are permutable with all other rules. The use of conjunctive superformulas allows us to get rid of non-conjunctive subformulas before the proof-search, unlike the dynamic use of permutabilities as in [Shankar 1992]. The use of conjunctive superformulas also covers all applications of *universal formulas* in the tableau frameworks proposed by Beckert and Hähnle [1992].

Subformula	least conjunctive superformula	set of ξ -names
$(\exists x(F(x) \wedge (B(x) \vee \exists yC(x, y))) \wedge \exists zD(z)) \vee E$	no	\emptyset
$\exists x(F(x) \wedge (B(x) \vee \exists yC(x, y))) \wedge \exists zD(z)$	no	\emptyset
$\exists x(F(x) \wedge (B(x) \vee \exists yC(x, y)))$	$\exists x(F(x) \wedge (B(x) \vee \exists yC(x, y)))$	$\{\mathcal{A}_1\}$
$F(x) \wedge (B(x) \vee \exists yC(x, y))$	$\exists x(F(x) \wedge (B(x) \vee \exists yC(x, y)))$	$\{\mathcal{A}_1\}$
$F(x)$	$F(x)$	$\{\mathcal{A}_2(x)\}$
$B(x) \vee \exists yC(x, y)$	$B(x) \vee \exists yC(x, y)$	$\{\mathcal{A}_3(x)\}$
$B(x)$	$B(x) \vee \exists yC(x, y)$	$\{\mathcal{A}_3(x)\}$
$\exists yC(x, y)$	$B(x) \vee \exists yC(x, y)$	$\{\mathcal{A}_3(x)\}$
$C(x, y)$	$B(x) \vee \exists yC(x, y)$	$\{\mathcal{A}_3(x)\}$
$\exists zD(z)$	$\exists zD(z)$	$\{\mathcal{A}_4\}$
$D(z)$	$\exists zD(z)$	$\{\mathcal{A}_4\}$
E	no	\emptyset

Table 3: Least conjunctive superformulas and sets of ξ -names of subformulas of the formula $\xi = (\exists x(F(x) \wedge (B(x) \vee \exists yC(x, y))) \wedge \exists zD(z)) \vee E$

2. x_1, \dots, x_m are all free variables of ξ_k in the order of their occurrences in ξ_k . If a ξ -name of a formula φ exists, then it is unique. Note that different formulas may have the same ξ -names. Also note that some subformulas of ξ do not have ξ -names (those who do not have a conjunctive superformula). We can use the *set of ξ -names* of a subformula. The set of ξ -names of a formula φ is either \emptyset or a singleton $\{\mathcal{A}_k(x_1, \dots, x_m)\}$. Table 3 illustrates sets of ξ -names.

Consider another example giving us the idea of an algorithm for assigning ξ -names. We will use this example for the illustration of all methods of this section.

8.1. EXAMPLE (ξ -names). Consider the formula of Example 1.3

$$\exists xyuv((a \approx b \supset g(x, u, v) \approx g(y, fc, fd)) \wedge (c \approx d \supset g(u, x, y) \approx g(v, fa, fb))).$$

Its negation normal form is

$$\xi = \exists xyuv((a \not\approx b \vee g(x, u, v) \approx g(y, fc, fd)) \wedge (c \not\approx d \vee g(u, x, y) \approx g(v, fa, fb))).$$

The only conjunctive subformulas of ξ are the subformulas $a \not\approx b \vee g(x, u, v) \approx g(y, fc, fd)$ and $c \not\approx d \vee g(u, x, y) \approx g(v, fa, fb)$. Thus, we can introduce the following ξ -names:

1. $\mathcal{A}_1(x, u, v, y)$ for all subformulas of $a \not\approx b \vee g(x, u, v) \approx g(y, fc, fd)$;
2. $\mathcal{A}_2(u, x, y, v)$ for all subformulas of $c \not\approx d \vee g(u, x, y) \approx g(v, fa, fb)$.

All other formulas have no ξ -names since they have no least conjunctive superformula.

In Section 7 we formalized basic superposition using ordering constraints. Here we will formalize it via *closures* used e.g. in [Bachmair et al. 1995].

- A *closure* is a pair $C \cdot \sigma$, where C is a clause and σ is a substitution. When σ is the empty substitution ε , it will sometimes be omitted.

Such a closure semantically corresponds to the clause $C\sigma$.

- Two closures $C_1 \cdot \sigma_1$ and $C_2 \cdot \sigma_2$ are called *variants* if C_1 is a variant of C_2 and $C_1\sigma_1$ is a variant of $C_2\sigma_2$.

We will identify variants.

The equality elimination method is based on transformations of closures which try to eliminate equality from them (hence the name “equality elimination”). The equality elimination procedure begins with the so-called initial closures and uses a basic superposition calculus described below.

- *Initial closures* are generated according to one of the three rules:

1. whenever a literal $s \not\approx t$ occurs in ξ and C is the set of ξ -names of this occurrence of $s \not\approx t$, the closure $s \approx t, C \cdot \varepsilon$ is an initial closure;
2. whenever the literal $s \approx t$ occurs in ξ and C is the set of ξ -names of this occurrence of $s \approx t$, the closure $s \not\approx t, C \cdot \varepsilon$ is an initial closure;
3. let literals $P(s_1, \dots, s_n)$ and $\neg P(t_1, \dots, t_n)$ occur in ξ and C_1, C_2 are their sets of ξ -names. Let the substitution ρ rename variables such that variables of $C_1\rho$ and C_2 are disjoint. Then the closure $s_1\rho \not\approx t_1, \dots, s_n\rho \not\approx t_n, C_1\rho, C_2 \cdot \varepsilon$ is an initial closure.

8.2. **EXAMPLE (Initial closures).** Let us continue Example 8.1. The formula ξ of that example has no predicate symbols different from \approx . Thus, the last case of the definition of initial closures is not applicable. The first two cases show how to associate an initial closure with every equality literal in ξ . Applying them, we obtain the following four initial closures (remember that we omit ε in closures):

1. $a \approx b, \mathcal{A}_1(x, u, v, y)$
2. $g(x, u, v) \not\approx g(y, fd, fd), \mathcal{A}_1(x, u, v, y)$
3. $c \approx d, \mathcal{A}_2(u, x, y, v)$
4. $g(u, x, y) \not\approx g(v, fa, fb), \mathcal{A}_2(u, x, y, v)$

We introduce several inference rules on closures defining a variant of basic superposition. We assume that the premises of the rules have disjoint variables, which can be achieved by renaming variables.

► *Basic (right and left) superposition rules* basic right superposition are the following inference rules:

$$\frac{s \approx t, C \cdot \sigma_1 \quad u[s'] \approx v, D \cdot \sigma_2}{u[t] \approx v, C, D \cdot \sigma_1 \sigma_2 \theta} \text{ (brs)} \quad \frac{s \approx t, C \cdot \sigma_1 \quad u[s'] \not\approx v, D \cdot \sigma_2}{u[t] \not\approx v, C, D \cdot \sigma_1 \sigma_2 \theta} \text{ (bls)}$$

where

1. θ is a most general unifier of $s\sigma_1$ and $s'\sigma_2$;
2. $t\sigma_1\theta \not\approx s\sigma_1\theta$ and $v\sigma_2\theta \not\approx u[s']\sigma_2\theta$;
3. s' is not a variable;
4. (for left superposition only) $u[s'] \not\approx v$ is the leftmost disequation in the second premise¹⁰.

► *Equality solution* is the following inference rule:

$$\frac{s \not\approx t, C \cdot \sigma}{C \cdot \sigma \text{mgu}(s\sigma, t\sigma)} \text{ (eqs)}$$

where $s \not\approx t$ is the leftmost disequation in the premise.

► **BS $_{\xi}$** is the inference system whose axioms are initial closures and whose inference rules are (bls), (brs) and (eqs).

► We call a *solution clause* any clause $C\sigma$ such that

1. $C \cdot \sigma$ is derivable in **BS $_{\xi}$** ;
2. C does not contain equality.

Let L_1, \dots, L_n be a solution clause. By replacing ξ -names to corresponding conjunctive subformulas, we obtain a multiset of formulas $\varphi_1, \dots, \varphi_n$ which we informally call *the formula image of the solution clause*.

8.3. **THEOREM.** *The following conditions are equivalent:*

¹⁰According to our definitions, a clause is a multiset of literals, so the use of the leftmost disequation is not quite correct, but this restriction can easily be formalized using the selection mechanism of Bachmair et al. [1995].

1. ξ is provable in first-order logic with equality;
2. There exist solution clauses C_1, \dots, C_m such that ξ is provable in the logical system whose axioms are the formula images of C_1, \dots, C_m and whose inference rules are rules of \mathbf{IK}_ξ except for (Ax) .

Thus, after obtaining a suitable set of solution clauses, we can reduce provability in logic with equality to provability in logic without equality and with axioms of a very special form. In subsequent sections, we will present two reformulations of this theorem which will be directly applicable to the tableau method and to the inverse method.

We illustrate the solution clauses by continuing Example 8.2 on page 672.

8.4. EXAMPLE (Solution clauses). Let us continue Example 8.1. The function signature for this example is $\mathcal{F} = \{a, b, c, d, f, g\}$. Consider the lexicographic path ordering \succ induced by the precedence relation $a \succ_{\mathcal{F}} b \succ_{\mathcal{F}} c \succ_{\mathcal{F}} d \succ_{\mathcal{F}} f \succ_{\mathcal{F}} g$. In this ordering, we have $a \succ b$ and $c \succ d$. Repeatedly applying all possible inference rules of \mathbf{BS}_ξ to the closures of Example 8.2, we obtain the following closures. The derivations are presented in the linear format, instead of tree-like derivations. In each line, the right column provides the information about the inference rule applied to obtain the closure at that line and the premises of the application of the inference rule.

5. $A_1(x, u, v, y) \cdot [y \mapsto x, u \mapsto fc, v \mapsto fd]$ (eqs) 2
6. $A_2(u, x, y, v) \cdot [v \mapsto u, x \mapsto fa, y \mapsto fb]$ (eqs) 4
7. $A_1(x_1, u_1, v_1, y_1), g(u_2, x_2, y_2) \not\approx g(v_2, fb, fb), A_2(u_2, x_2, y_2, v_2)$ (bls) 1, 4
8. $A_2(u_1, x_1, y_1, v_1), g(x_2, u_2, v_2) \not\approx g(y_2, fd, fd), A_1(x_2, u_2, v_2, y_2)$ (bls) 3, 2
9. $A_1(x_1, u_1, v_1, y_1), A_2(u_2, x_2, y_2, v_2) \cdot [v_2 \mapsto u_2, x_2 \mapsto fb, y_2 \mapsto fb]$ (eqs) 7
10. $A_2(u_1, x_1, y_1, v_1), A_1(x_2, u_2, v_2, y_2) \cdot [y_2 \mapsto x_2, u_2 \mapsto fd, v_2 \mapsto fd]$ (eqs) 8

No further applications of the inference rules of \mathbf{BS}_ξ are possible. Thus, we obtain the following four solutions clauses obtained from closures 5, 6, 9, and 10, respectively:

- 5' $A_1(x, fc, fd, x)$
- 6' $A_2(u, fa, fb, u)$
- 9' $A_1(x_1, u_1, v_1, y_1), A_2(u_2, fb, fb, u_2)$
- 10' $A_2(u_1, x_1, y_1, v_1), A_1(x_2, fd, fd, x_2)$

We do not give a complete derivation of the goal formula using these solution clauses now, but rather postpone it to the next sections where such examples will be given both for a tableau calculus \mathbf{T}_ξ and for an inverse method \mathbf{I}_ξ . Conceptually, the generation of the solution is independent of their use in a particular method. The integration of the solution clause generation into a particular method in an “efficient” way is a separate matter not discussed here.

8.2. Equality elimination for semantic tableaux

In the previous section we have introduced ξ -names in order to deal with occurrences of subformulas instead of subformulas themselves. In this section, we modify the free-variable tableau calculus to deal with ξ -names instead of formulas. The new tableaux will be called ξ -tableaux.

- ▶ A ξ -branch is any finite multiset of atoms of the form $\mathcal{A}_k(t_1, \dots, t_m)$ such that $\mathcal{A}_k(x_1, \dots, x_m)$ is a ξ -name of some subformula of ξ .
- ▶ A ξ -tableau is any finite multiset $\{C_1, \dots, C_n\}$ of ξ -branches, denoted by $C_1 \mid \dots \mid C_n$.

Our calculus for ξ -tableaux uses *subset unification*.

- ▶ A substitution σ is called a *subset unifier* of a clause C_1 against a clause C_2 if $C_1\sigma \subseteq C_2\sigma$.
- ▶ A subset unifier of C_1 against C_2 is called *minimal* if it is minimal w.r.t. \leq among all subset unifiers of C_1 against C_2 .

Subset-unification has been earlier used in SLO-resolution by the name of *θ -subsumption* in [Minker, Rajasekar and Lobo 1991]. Subset-unifiability is NP-complete [Degtyarev and Voronkov 1994a]. Some other properties of subset unification are discussed in [Degtyarev and Voronkov 1994a].

On ξ -tableaux, we will introduce a calculus called T_ξ which consists of two inference rules: tableau expansion and branch closure.

- ▶ Let C_1, C_2 and C be the sets of ξ -names of φ, ψ and $\varphi \wedge \psi$, where $\varphi \wedge \psi$ is a subformula of ξ . Then the following is a ξ -tableau expansion rule

$$\frac{D_1 \mid D_2 \mid \dots \mid D_m}{(C_1, D_1 \mid C_2, D_1 \mid D_2 \mid \dots \mid D_m)\sigma} (te_\xi)$$

where σ is a minimal subset unifier of C against D_1 . We assume that the variables of the premise are disjoint from the variables of C, C_1, C_2 which can be achieved by renaming variables in the tableau.

This rule can be described in a more simple (but less compact) form, when we consider its two cases depending on the number of elements in C . There are two possibilities: C is either \emptyset or a singleton multiset $\{\mathcal{A}_i(\bar{x})\}$. As for C_1 and C_2 , they are nonempty since φ and ψ are conjunctive subformulas of $\varphi \wedge \psi$. Thus, $C_1 = \{\mathcal{A}_j(\bar{y})\}$ and $C_2 = \{\mathcal{A}_k(\bar{z})\}$.

1. If $C = \emptyset$, then the rule (te_ξ) becomes

$$\frac{D_1 \mid D_2 \mid \dots \mid D_m}{D_1, \mathcal{A}_j(\bar{y}) \mid D_1, \mathcal{A}_k(\bar{z}) \mid D_2 \mid \dots \mid D_m} (te_\xi)$$

2. If $C = \{\mathcal{A}_i(\bar{x})\}$, then the rule (te_ξ) becomes

$$\frac{D_1, \mathcal{A}_i(\bar{t}) \mid D_2 \mid \dots \mid D_m}{D_1, \mathcal{A}_j(\bar{y})\sigma \mid D_1, \mathcal{A}_k(\bar{z})\sigma \mid D_2 \mid \dots \mid D_m} (te_\xi)$$

where $\sigma = [\bar{x} \mapsto \bar{t}]$.

Note that in the latter rule the variables \bar{x} do not occur in the premise but may occur in \bar{y}, \bar{z} . Therefore, it is enough to apply σ only to $\mathcal{A}_j(\bar{y}), \mathcal{A}_k(\bar{z})$.

This rule corresponds to the β -rule

$$\frac{\beta}{\beta_1 \mid \beta_2}$$

of Smullyan [1968] modified for ξ -branches.

► Let C be any solution clause. Then the following is a ξ -branch closure rule

$$\frac{C_1 \mid C_2 \mid \dots \mid C_m}{(C_2 \mid \dots \mid C_m)\sigma} (bc_\xi)$$

where σ is a minimal subset-unifier of C against C_1 . We assume that the variables of the tableau are disjoint from the variables of C which can be achieved by renaming variables in C .

► The calculus \mathbf{T}_ξ is the calculus whose axiom is the tableau consisting of one branch \square and whose inference rules are (te_ξ) and (bc_ξ) .

Due to the use of least conjunctive superformulas, we do not need any analogues of α - or γ -rules in \mathbf{T}_ξ . The ξ -tableau expansion rule is an analogue of the β -rule. The calculus \mathbf{T}_ξ was introduced in [Degtyarev and Voronkov 1996b] in a slightly different form where both closures and ξ -tableaux were derivable objects. In fact, the calculus T_ξ of [Degtyarev and Voronkov 1996b] is the union of \mathbf{T}_ξ of this chapter and \mathbf{BS}_ξ .

8.5. THEOREM (Equality elimination, ξ -tableau formulation). *The following conditions are equivalent:*

1. ξ is provable in first-order logic with equality;
2. the empty tableau $\#$ is derivable in \mathbf{T}_ξ .

This theorem can also be reformulated as a refinement of Theorem 3.1 (Herbrand's theorem for free variable tableaux):

8.6. THEOREM. *The following conditions are equivalent:*

1. ξ is provable in first-order logic with equality;
2. there is a ξ -tableau \mathcal{T} constructed from \square by applications of the ξ -tableau expansion rule and a substitution θ such that every branch in $\mathcal{T}\theta$ contains an instance of a solution clause.

Theorem 3.1 on page 639 just asserted that we have to search for a substitution making all branches inconsistent, which leads to simultaneous rigid E -unification. Theorem 8.6 contains a more constructive statement: branches can be closed by instances of solution clauses.

Let us illustrate equality elimination for tableaux.

8.7. **EXAMPLE.** Consider again Examples 8.1 and 8.4. For the formula ξ of these examples, we get the following ξ -tableau expansion rule:

$$\frac{D_1 \mid D_2 \mid \dots \mid D_m}{A_1(x, u, v, y), D_1 \mid A_2(u, x, y, v), D_1 \mid D_2 \mid \dots \mid D_m} (te_\xi)$$

where x, y, u, v are variables not occurring in the premise of this tableau.

For the branch closure rules, we will need solution clauses 5', 6', 9' and 10' of Example 8.4. The derivation is as follows:

1. \square
2. $A_1(x_0, u_0, v_0, y_0) \mid A_2(u_0, x_0, y_0, v_0)$ (te_ξ), 1
3. $A_1(x_0, u_0, v_0, y_0), A_1(x_1, u_1, v_1, y_1) \mid$ (te_ξ), 2
 $A_1(x_0, u_0, v_0, y_0), A_2(u_1, x_1, y_1, v_1) \mid A_2(u_0, x_0, y_0, v_0)$
4. $A_1(x_0, u_0, v_0, y_0), A_2(fc, x_1, x_1, fd) \mid A_2(u_0, x_0, y_0, v_0)$ (bc_ξ), 3
5. $A_2(fd, u_0, u_0, fd)$ (bc_ξ), 4
6. $A_2(fd, u_0, u_0, fd), A_1(x_2, u_2, v_2, y_2) \mid$ (te_ξ), 5
 $A_2(fd, u_0, u_0, fd), A_2(u_2, x_2, y_2, v_2)$
7. $A_2(fd, fb, fb, fd), A_2(u_2, x_2, y_2, v_2)$ (bc_ξ), 6
8. $\#$ (bc_ξ), 7

Here the branch closure rules used to obtain tableaux 4, 5, 7, 8 have been applied using solution clauses 5', 10', 9', 6', respectively.

A similar idea: combination of proof-search in tableaux and a bottom-up equality saturation of the original formula, is used in [Moser et al. 1995] for constructing a model elimination tableau with refined paramodulation. The main difference between the two approaches is that the elimination of equality in the tableau part in [Moser et al. 1995] is partial but not total. This requires to introduce in the tableau part *relaxed paramodulation* of Snyder and Lynch [1991] and even come back to *lazy paramodulation* of Gallier and Snyder [1989] dropping *top unification* from relaxed paramodulation. Comparing the two forms of paramodulation Snyder and Lynch [1991] write: "Our original formulation of RPC (relaxed paramodulation calculus) used the form of lazy paramodulation from [Gallier and Snyder 1989], but clearly the refinement to top unification is superior, since it restricts the number of inferences and conceptually it clarifies the dividing line between unification and completely lazy unification".

8.3. Equality elimination for the inverse method

Equality elimination for the inverse method is based on the same characterization of provability in terms of solution clauses as that for the tableau method (Theorem 8.6). However, the resulting calculi for the inverse method and for the tableau method are quite different because the inverse method is local while the tableau

method is global. Among other inference rules, the inverse method uses the *factoring rule* that is absent in the tableau method. In a certain sense, equality elimination for tableaux and the inverse method are dual to each other which reflects the duality between bottom-up and top-down reasoning.

We will formulate the inverse method in the form of a logical calculus I_ξ over ξ -clauses.

- A ξ -clause is any finite multiset of atoms of the form $\mathcal{A}_k(t_1, \dots, t_m)$ such that $\mathcal{A}_k(x_1, \dots, x_m)$ is a ξ -name of some subformula of ξ .

The definition of ξ -clauses is the same as that of ξ -branches. We use ξ -clauses to emphasize, first, that they do not belong to particular tableaux and second, that the calculus of ξ -clauses is similar to the resolution calculus. The calculus I_ξ consists of two inference rules: the conjunction rule and the factoring rule.

- Let C_1, C_2 and C be the sets of ξ -names of φ, ψ and $\varphi \wedge \psi$, where $\varphi \wedge \psi$ is a subformula of ξ and D_1, D_2 be clauses. Let \bar{x} be all variables of $\varphi \wedge \psi$. Then the following is a ξ -conjunction rule

$$\frac{D_1, C_1[\bar{x} \mapsto \bar{s}] \quad D_2, C_2[\bar{x} \mapsto \bar{t}]}{(D_1, D_2, C[\bar{x} \mapsto \bar{s}])\text{mgu}(\bar{s}, \bar{t})} (\wedge_\xi)$$

- The following is a *factoring rule*

$$\frac{C, A, B}{(C, A)\text{mgu}(A, B)} (fac)$$

- The calculus I_ξ is the calculus whose axioms are solution clauses and whose inference rules are (\wedge_ξ) and (fac) .

8.8. THEOREM. *The following conditions are equivalent:*

1. ξ is provable in first-order logic with equality;
2. the empty clause \square is derivable in I_ξ .

8.9. EXAMPLE. Consider again Examples 8.1 and 8.4. For the formula ξ of these examples, we get the following conjunction rule:

$$\frac{D_1, \mathcal{A}_1(\bar{s}_x, s_u, s_v, s_y) \quad D_2, \mathcal{A}_2(t_u, t_x, t_y, t_v)}{(D_1, D_2)\text{mgu}(\langle s_x, s_y, s_u, s_v \rangle, \langle t_x, t_y, t_u, t_v \rangle)} (\wedge_\xi)$$

We start with solution clauses 5', 6', 9' and 10' of Example 8.4. The derivation is as follows:

- 5'. $\mathcal{A}_1(x, fc, fd, x)$
- 6'. $\mathcal{A}_2(u, fa, fb, u)$
- 9'. $\mathcal{A}_1(x_1, u_1, v_1, y_1), \mathcal{A}_2(u_2, fb, fb, u_2)$
- 10'. $\mathcal{A}_2(u_3, x_3, y_3, v_3), \mathcal{A}_1(x_4, fd, fd, x_4)$
1. $\mathcal{A}_1(x_5, u_5, v_5, y_5), \mathcal{A}_2(u_6, x_6, y_6, v_6)$ $(\wedge_\xi), 9', 10'$
2. $\mathcal{A}_1(x_7, u_7, v_7, y_7)$ $(\wedge_\xi), 5', 1$
3. \square $(\wedge_\xi), 6', 2$

Note that the same derivation is obtained by positive hyperresolution, when we replace the ξ -conjunction rule of this example by the clause

$$\neg \mathcal{A}_1(x, u, v, y), \neg \mathcal{A}_2(u, x, y, v).$$

8.4. Further issues in equality elimination

Deduction in the calculus \mathbf{BS}_ξ can be improved by redundancy deletion. Besides tautology deletion, we can use such deletion strategies as *basic subsumption*, *basic simplification*, and *basic blocking* [Bachmair et al. 1992, Bachmair et al. 1995].

Application of arbitrary simplification in the basic setting may yield incompleteness, as we can see from the following example taken from [Degtyarev 1985]. Let S consist of three closures

1. $f_1x \approx f_2x$;
2. $f_3c \approx c$;
3. $f_1f_3y \not\approx f_2c$.

We use any reduction ordering such that $f_1x \succ f_2x$. This set of closures is inconsistent, and there is a derivation of the empty closure from 1–3 by basic superposition. We can simplify closure 3 by 1 obtaining

$$4. f_2x \not\approx f_2c \cdot [x \mapsto f_3y].$$

It is easy to see that it is impossible to derive the empty closure from 1, 2, 4 by basic superposition.

If we use basic simplification instead, the simplification would give us the closure

$$4. f_2f_3y \not\approx f_2c \cdot \varepsilon.$$

The applications of equality elimination to logic programming [Degtyarev and Voronkov 1995c, Degtyarev and Voronkov 1996d], to the inverse method [Degtyarev and Voronkov 1995a], and to semantic tableaux [Degtyarev and Voronkov 1995b, Degtyarev and Voronkov 1996b] can be considered as a transformation of a set of clauses with equality into a set of clauses without equality. However, all these applications share the same problem: the transformation itself and the resulting set of clauses without equality can, in general, be infinite (since there may be an infinite number of solution clauses). In order to cope with problem, Degtyarev, Koval and Voronkov [1995] and Degtyarev and Voronkov [1996c] introduce a new inference rule: the so-called *basic folding* for logic programs with equality. By introduction of new predicate symbols encoding partially solved equations we ensure termination of the equality elimination procedure, thus allowing one to obtain elegant finite programs without equality from equational logic programs. Moreover, the resulting program without equality can sometimes give a unique solution from an infinite

number of solutions to the original program. The basic folding is a promising approach in the area of machine learning because it gives us a possibility to automatically discover new notions and to reformulate a logical theory in terms of the new notions.

The essential difference between basic folding and Brand's [1975] modification method is that we try to *solve* all equations in the bodies by specialized applications of equality rules: basic superposition and equality solution. It allows us to automatically "subsume" the axioms of symmetry and transitivity, and also to essentially decrease the number of logical consequences due to the use of orderings. According to the Brand's method, these axioms must be explicitly applied to all positive occurrences of the predicate \approx . In [Degtyarev and Voronkov 1996c, Degtyarev and Voronkov 2001a] we show that basic folding may give a considerably simpler set of clauses: in an example of that paper the basic folding generates a set of clauses having a finite SLD-tree, while Brand's transformation generates a logic program with an infinite SLD-tree. However, the basic folding is currently only defined for sets of Horn clauses.

9. Equality reasoning in nonclassical logics

Equality handling in nonclassical logics is more problematic than in the classical one. Whereas problems in handling equality in tableaux for classical logic have been caused by the presence of rigid variables and do not appear in resolution-based theorem proving, the picture for nonclassical logics is entirely different. In this section we demonstrate that handling equality in intuitionistic logic necessarily requires the use of simultaneous rigid E -unification. We also demonstrate that the same is true for some formalizations of modal logics with equality.

9.1. Intuitionistic logic with equality

The material of this section is mainly based on [Voronkov 1996a, Voronkov 1996d, Voronkov 1998c].

For technical reasons known to anyone with background in intuitionistic logic, in this section we change the definition of a sequent.

- A *sequent* is an expression $\Gamma \rightarrow \Delta$, where Γ, Δ are sequences of formulas and Δ contains at most one formula. Γ (respectively Δ) is called the *antecedent* (respectively, the *succedent*) of this sequent.

The condition on the succedent to contain one formula is essential and leads to intuitionistic logic instead of classical.

Thus obtained *sequent calculus* $\mathbf{LJ}^=$ for intuitionistic logic with equality is shown in Figure 14. This calculus derives intuitionistic sequents. In fact, the rules of $\mathbf{LJ}^=$ are precisely the rules of $\mathbf{LK}^=$ adapted to meet the new definition of sequents.

This chapter is not intended to serve as an introduction to intuitionistic logic, so we will only assert some most essential properties of $\mathbf{LJ}^=$ needed to understand

$$\begin{array}{c}
\frac{}{\Gamma, A, \Delta \rightarrow A} (Ax) \qquad \frac{}{\Gamma \rightarrow t \approx t} (refl) \\
\\
\frac{\Gamma[x \mapsto t], s \approx t \rightarrow \Delta[x \mapsto t]}{\Gamma[x \mapsto s], s \approx t \rightarrow \Delta[x \mapsto s]} (\approx) \\
\\
\frac{\Gamma, \varphi, \psi, \Xi \rightarrow \Delta}{\Gamma, \varphi \wedge \psi, \Xi \rightarrow \Delta} (\wedge \rightarrow) \qquad \frac{\Gamma \rightarrow \varphi \quad \Gamma \rightarrow \psi}{\Gamma \rightarrow \varphi \wedge \psi} (\rightarrow \wedge) \\
\\
\frac{\Gamma, \varphi, \Xi \rightarrow \Delta \quad \Gamma, \psi \rightarrow \Delta}{\Gamma, \varphi \vee \psi, \Xi \rightarrow \Delta} (\vee \rightarrow) \\
\\
\frac{\Gamma \rightarrow \varphi}{\Gamma \rightarrow \varphi \vee \psi} (\rightarrow \vee_1) \qquad \frac{\Gamma \rightarrow \psi}{\Gamma \rightarrow \varphi \vee \psi} (\rightarrow \vee_2) \\
\\
\frac{\Gamma, \psi, \Xi \rightarrow \Delta \quad \Gamma, \varphi \supset \psi, \Xi \rightarrow \varphi}{\Gamma, \varphi \supset \psi, \Xi \rightarrow \Delta} (\supset \rightarrow) \qquad \frac{\varphi, \Gamma \rightarrow \psi}{\Gamma \rightarrow \varphi \supset \psi} (\rightarrow \supset) \\
\\
\frac{\Gamma, \neg \varphi \rightarrow \varphi}{\Gamma, \neg \varphi, \Xi \rightarrow} (\neg \rightarrow) \qquad \frac{\varphi, \Gamma \rightarrow}{\Gamma \rightarrow \neg \varphi} (\rightarrow \neg) \\
\\
\frac{\Gamma, \varphi[x \mapsto t], \forall x \varphi, \Xi \rightarrow \Delta}{\Gamma, \forall x \varphi, \Xi \rightarrow \Delta} (\forall \rightarrow) \qquad \frac{\Gamma \rightarrow \varphi[x \mapsto y]}{\Gamma \rightarrow \forall x \varphi} (\rightarrow \forall) \\
\\
\frac{\Gamma, \varphi[x \mapsto y], \Xi \rightarrow \Delta}{\Gamma, \exists x \varphi, \Xi \rightarrow \Delta} (\exists \rightarrow) \qquad \frac{\Gamma \rightarrow \varphi[x \mapsto t]}{\Gamma \rightarrow \exists x \varphi} (\rightarrow \exists)
\end{array}$$

The rules $(\rightarrow \forall)$ and $(\exists \rightarrow)$ satisfy the standard eigenvariable conditions.

Figure 14: Calculus $\mathbf{LJ}^=$

the problems arising in handling equality. *Regular derivations* in $\mathbf{LJ}^=$ are defined in the same way as for $\mathbf{LK}^=$ (page 624).

As in the case with $\mathbf{LK}^=$, regular derivations are enough:

9.1. THEOREM. *Any sequent derivable in $\mathbf{LJ}^=$ has a regular derivation.*

Two example derivations in $\mathbf{LJ}^=$ of the formula

$$\forall x(x \approx a \vee x \approx b \supset \exists y(f(x, y) \approx b)) \supset \exists u \exists v \exists w(f(f(u, v), w) \approx b)$$

are given in Figures 15 and 16. The first derivation is regular while the second is not regular.

Unlike $\mathbf{LK}^=$, the calculus $\mathbf{LJ}^=$ has the *explicit definability property* which can be expressed by

In this example, we denote by φ the formula $\forall x(x \approx a \vee x \approx b \supset \exists y(f(x, y) \approx b))$. We also denote by \dots some irrelevant parts of the sequents.

$$\begin{array}{c}
 \frac{\frac{\frac{f(a, y) \approx b, f(b, z) \approx b, \varphi \rightarrow b \approx b}{f(a, y) \approx b, f(b, z) \approx b, \varphi \rightarrow f(b, z) \approx b} (\text{refl})}{f(a, y) \approx b, f(b, z) \approx b, \varphi \rightarrow \exists w(f(b, w) \approx b)} (\rightarrow \exists) \\
 \frac{f(a, y) \approx b, \exists y(f(b, y) \approx b), \varphi \rightarrow \exists w(f(b, w) \approx b)}{f(a, y) \approx b, b \approx a \vee b \approx b \supset \exists y(f(b, y) \approx b), \varphi \rightarrow \exists w(f(b, w) \approx b)} (\rightarrow \vee_2) \\
 \frac{f(a, y) \approx b, b \approx a \vee b \approx b \supset \exists y(f(b, y) \approx b), \varphi \rightarrow \exists w(f(b, w) \approx b)}{f(a, y) \approx b, \varphi \rightarrow \exists w(f(b, w) \approx b)} (\vee \rightarrow) \\
 \frac{\frac{\frac{f(a, y) \approx b, \varphi \rightarrow \exists w(f(f(a, y), w) \approx b)}{f(a, y) \approx b, \varphi \rightarrow \exists v \exists w(f(f(a, v), w) \approx b)} (\rightarrow \exists)}{\exists y(f(a, y) \approx b), \varphi \rightarrow \exists v \exists w(f(f(a, v), w) \approx b)} (\exists \rightarrow) \\
 \frac{\exists y(f(a, y) \approx b), \varphi \rightarrow \exists v \exists w(f(f(a, v), w) \approx b)}{a \approx a \vee a \approx b \supset \exists y(f(a, y) \approx b), \varphi \rightarrow \exists v \exists w(f(f(a, v), w) \approx b)} (\rightarrow \vee_1) \\
 \frac{a \approx a \vee a \approx b \supset \exists y(f(a, y) \approx b), \varphi \rightarrow \exists v \exists w(f(f(a, v), w) \approx b)}{\varphi \rightarrow \exists v \exists w(f(f(a, v), w) \approx b)} (\vee \rightarrow) \\
 \frac{\varphi \rightarrow \exists v \exists w(f(f(a, v), w) \approx b)}{\varphi \rightarrow \exists u \exists v \exists w(f(f(u, v), w) \approx b)} (\rightarrow \exists) \\
 \frac{\varphi \rightarrow \exists u \exists v \exists w(f(f(u, v), w) \approx b)}{\rightarrow \varphi \supset \exists u \exists v \exists w(f(f(u, v), w) \approx b)} (\rightarrow \supset)
 \end{array}$$

Figure 16: An irregular derivation in \mathbf{LJ}^-

9.2. THEOREM. A sequent $\rightarrow \exists x\varphi$ is derivable in $\mathbf{LJ}^=$ if and only if there is a term t such that $\rightarrow \varphi[x \mapsto t]$ is derivable in $\mathbf{LJ}^=$.

Unlike $\mathbf{LK}^=$, in $\mathbf{LJ}^=$ we cannot get rid of δ -rules, since there are no analogues of skolemization.

9.2. Simultaneous rigid E -unification is inevitable

We introduce one technical notion.

- An inference rule is called *invertible* if the derivability of the conclusion of the rule implies the derivability of all premises of the rule.

All rules in $\mathbf{LK}^=$ are invertible, while $\mathbf{LJ}^=$ has some non-invertible rules.

All the methods considered in this chapter and complete for first-order classical logic with equality do not work for intuitionistic logic. The first thing we can note is that simultaneous rigid E -unification has a natural representation in intuitionistic logic in the following way.

Consider any system R of rigid equations defined by

$$R = \begin{array}{ccc} s_{11} \approx t_{11} & s_{1n_1} \approx t_{1n_1} & \vdash_{\forall} \quad s'_1 \approx t'_1 \\ \vdots & \vdots & \vdots \\ s_{k1} \approx t_{k1} & \dots & s_{kn_k} \approx t_{kn_k} \quad \vdash_{\forall} \quad s'_k \approx t'_k \end{array} \quad (9.1)$$

Let x_1, \dots, x_m be all variables occurring in R . Define the formulas ψ_R and φ_R as follows:

$$\begin{aligned} \psi_R = & (s_{11} \approx t_{11} \wedge \dots \wedge s_{1n_1} \approx t_{1n_1} \supset s'_1 \approx t'_1) \quad \wedge \\ & \dots \quad \wedge \\ & (s_{k1} \approx t_{k1} \wedge \dots \wedge s_{kn_k} \approx t_{kn_k} \supset s'_k \approx t'_k) \end{aligned} \quad (9.2)$$

$$\varphi_R = \exists x_1 \dots \exists x_m \psi_R \quad (9.3)$$

By considering all possible derivations of φ_R in $\mathbf{LJ}^=$, or by using Theorem 9.2, it is easy to see that φ_R is derivable in $\mathbf{LJ}^=$ if and only if there is a substitution σ such that the formula $\psi_R\sigma$ is derivable in $\mathbf{LJ}^=$. Using the fact that the rules $(\rightarrow \wedge)$, $(\rightarrow \supset)$ and $(\wedge \rightarrow)$ are invertible we can prove that $\psi_R\sigma$ is derivable in $\mathbf{LJ}^=$ if and only if for all $j \in \{1, \dots, k\}$ the sequent

$$s_{j1}\sigma \approx t_{j1}\sigma, \dots, s_{jn_j}\sigma \approx t_{jn_j}\sigma \rightarrow s'_j\sigma \approx t'_j\sigma$$

is derivable in $\mathbf{LJ}^=$. Derivability of such sequents in $\mathbf{LJ}^=$ is equivalent to their derivability in $\mathbf{LK}^=$. Hence, ψ is provable if and only if σ is a solution to R . Therefore, any proof of φ_R gives us a solution to system of rigid equations (9.1). This implies that handling simultaneous rigid E -unification is “inevitable” for designing complete algorithms for intuitionistic logic with equality: every complete semi-decision

algorithm that builds a proof in $\mathbf{LJ}^=$, applied to φ_R would build a solution to R . This fact was observed in [Voronkov 1996a, Voronkov 1996d]. Thus, unlike classical logic, handling simultaneous rigid E -unification is inevitable in any automated reasoning system for intuitionistic logic with equality.

9.3. Automated reasoning modulo simultaneous rigid E -unification

We established that any complete system of automated reasoning for intuitionistic logic gives us a semi-decision algorithm for simultaneous rigid E -unification. In this section, we show how one can design semi-decision algorithms for $\mathbf{LJ}^=$ modulo simultaneous rigid E -unification. This means that we assume that a semi-decision procedure P for simultaneous rigid E -unification is given and we try to give a general semi-decision procedure for $\mathbf{LJ}^=$ which may call P from time to time. The idea of such a procedure can already be found in equational matings [Gallier et al. 1987, Gallier et al. 1988, Gallier et al. 1992]. However, for intuitionistic logic we have to elaborate several things due to the existence of non-invertible rules and to the presence of δ -rules. Due to non-invertible rules, we will use *derivation skeletons* instead of tableaux or matrices. We will also built-in simultaneous rigid E -unification directly into the calculus by means of *constraints*. (These constraints are different from the equality and ordering constraints considered so far.) The exposition here mainly follows [Voronkov 1996d, Voronkov 1998c].

► A *derivation skeleton* is a tree such that

1. its nodes are labelled by the names of inference rules in $\mathbf{LJ}^=$, except for (\approx) ;
2. the number of parents of a node labelled by a name of an inference rule is equal to the number of the premises in such a rule in $\mathbf{LJ}^=$;
3. all nodes labelled by antecedent rules or (Ax) are additionally labelled by a natural number.

The natural number in an antecedent rule or in an axiom (Ax) represents the index of the main formula (counting from 0) of this rule in the antecedent of the conclusion of the rule. In other words, this number is the number of formulas in Γ in the corresponding rules of $\mathbf{LJ}^=$ (Figure 14). We always display the number as an index.

► The *skeleton* of a derivation Π is obtained from Π by removing all sequents and all applications of rules (\approx) , and by adding corresponding indices for antecedent rules and axioms (Ax) .

We display derivation skeletons as derivations with omitted sequents, nodes are denoted by horizontal bars with labels displayed to the right of the bar. For example, both derivations shown in Figures 15 and 16 have the same skeleton

$$\begin{array}{c}
\text{---} \quad (refl) \\
\text{---} \quad (\rightarrow \exists) \quad \text{---} \quad (refl) \\
\text{---} \quad (\exists \rightarrow)_1 \quad \text{---} \quad (\rightarrow \vee_2) \\
\hline
\text{---} \quad (\supset \rightarrow)_1 \\
\quad \text{---} \quad (\forall \rightarrow)_1 \\
\quad \text{---} \quad (\rightarrow \exists) \quad \text{---} \quad (refl) \\
\quad \text{---} \quad (\exists \rightarrow)_0 \quad \text{---} \quad (\rightarrow \vee_1) \\
\quad \hline
\quad \text{---} \quad (\supset \rightarrow)_0 \\
\quad \quad \text{---} \quad (\forall \rightarrow)_0 \\
\quad \quad \text{---} \quad (\rightarrow \exists) \\
\quad \quad \text{---} \quad (\rightarrow \supset)
\end{array}$$

Derivation skeletons are abstractions of derivations that encode information about the structures of derivations. They abstract from the following information:

1. instantiation of variables in quantifier rules;
2. the order of applications of the equality rules (\approx) and (\approx).

A problem similar to the existence of equational mating in terminology of Gallier et al. [1992] is the following

- *skeleton instantiation problem*: given a sequent \mathcal{S} and a skeleton S , does there exist a derivation of \mathcal{S} in $\mathbf{LJ}^=$ with the skeleton S .

For formulas without equality this problem is decidable in polynomial-time [Voronkov 1996c, Voronkov 2001b]. For formulas with equality the skeleton instantiation problem is undecidable. In fact, we have [see Voronkov 1996d, Voronkov 1998c]:

9.3. THEOREM. *Simultaneous rigid E -unifiability is polynomial-time equivalent to the skeleton instantiation problem.*

A more precise formulation of this statement with respect to signatures may be found in [Voronkov 1998d, Voronkov 1999].

9.3.1. Constraints

As in the case with classical tableaux, we have to make a step from $\mathbf{LJ}^=$ to a free variable calculus. As it has been show above, handling simultaneous rigid E -unification is inevitable for free-variable versions of $\mathbf{LJ}^=$. What we will do here is to introduce simultaneous rigid E -unification directly in the calculus in the form of *constraints*. This trick is similar to the introduction of constraints in the tableau calculus made in Section 7.2, but the notion of a constraint will be different. Here, constraints will encode the information on possible instantiations of skeletons to real derivations.

Based on the constraints, we define the constraint calculus $\mathbf{LJ}_c^=$. This calculus can be used for defining sequent style proof procedures for intuitionistic logic with

equality. The proof procedures based on \mathbf{LJ}_c^- consist of two parts: the construction of a derivation skeleton and the constraint satisfaction part. The constraint to be satisfied is computed from the skeleton of a derivation. The construction of a skeleton can also be considered as a sequence of applications of tableau expansion rules.

► *Constraints* are defined inductively as follows:

1. \top is a constraint;
2. For any terms t, t_1, \dots, t_n , $t \not\approx \{t_1, \dots, t_n\}$ is a constraint;
3. Any rigid equation $s_1 \approx t_1, \dots, s_n \approx t_n \vdash_V s \approx t$ is a constraint;
4. If C_1, C_2 are constraints, then $C_1 \wedge C_2$ is a constraint;
5. If C is a constraint, x is variable, then $\exists x C$ is a constraint.

Constraints can be regarded as first-order formulas using the atomic formulas \top , $t \not\approx \{t_1, \dots, t_n\}$ and $E \vdash_V s \approx t$ such that $\dots \not\approx \{\dots\}$ and $\dots \vdash_V \dots$ are considered as a family of predicate symbols of arbitrary arities. Then, we can speak about

► the set $\text{vars}(C)$ of *free variables* of a constraint C .

Constraints describe the domain of substitutions. If a constraint C is true on a substitution σ we say that σ *satisfies* C :

► The notion *a substitution σ satisfies the constraint C* , denoted $\sigma \models C$, is defined inductively as follows:

1. $\sigma \models \top$ for every substitution σ ;
2. $\sigma \models E \vdash_V s \approx t$ if σ is a solution to the rigid equation $E \vdash_V s \approx t$;
3. $\sigma \models t \not\approx \{t_1, \dots, t_n\}$ if $t\sigma$ is a variable and $t\sigma$ does not occur in $t_1\sigma, \dots, t_n\sigma$;
4. $\sigma \models C_1 \wedge C_2$ if $\sigma \models C_1$ and $\sigma \models C_2$;
5. $\sigma \models \exists x C$ if there is a substitution τ such that $\tau \models C$ and $y\tau = y\sigma$ for every variable y different from x .

► Constraints C_1 and C_2 are *equivalent* if for every substitution σ we have $\sigma \models C_1$ if and only if $\sigma \models C_2$.

► A constraint C is *satisfiable* if there is a substitution σ satisfying C .

9.3.2. Free variable calculus \mathbf{LJ}_c^-

In this section we introduce the calculus \mathbf{LJ}_c^- of *constrained sequents*. Before giving its inference rules, we introduce the following notation.

► For a multiset of formulas Γ , denote by Γ_{\approx} the multiset of formulas $\{s \approx t \mid s \approx t \in \Gamma\}$.

The calculus \mathbf{LJ}_c^- deals with *constraint sequents*:

► a *constrained sequent* is an expression $S \cdot C$ where S is a sequent and C is a constraint;

► the *constraint calculus* \mathbf{LJ}_c^- of constrained sequents is shown in Figure 17.

The notion of a skeleton for \mathbf{LJ}_c^- -derivations is similar to that of \mathbf{LJ}^- -derivations.

Consider two examples of derivations in \mathbf{LJ}_c^- . In both examples we derive the same formula as in Figures 15 and 16, augmented with some constraint. In the first

$$\begin{array}{c}
\frac{}{\Gamma, A, \Delta \rightarrow A \cdot (\Gamma_{\approx} \cup \Delta_{\approx} \vdash_{\forall} s_1 \approx t_1 \wedge \dots \wedge \Gamma_{\approx} \cup \Delta_{\approx} \vdash_{\forall} s_n \approx t_n)} (Ax) \\
\\
\frac{}{\Gamma \rightarrow t \approx t \cdot (\Gamma_{\approx} \vdash_{\forall} s \approx t)} (refl) \\
\\
\frac{\Gamma, \varphi, \psi, \Xi \rightarrow \Delta \cdot \mathcal{C}}{\Gamma, \varphi \wedge \psi, \Xi \rightarrow \Delta \cdot \mathcal{C}} (\wedge \rightarrow) \qquad \frac{\Gamma \rightarrow \varphi \cdot \mathcal{C}_1 \quad \Gamma \rightarrow \psi \cdot \mathcal{C}_2}{\Gamma \rightarrow \varphi \wedge \psi \cdot (\mathcal{C}_1 \wedge \mathcal{C}_2)} (\rightarrow \wedge) \\
\\
\frac{\Gamma, \varphi, \Xi \rightarrow \Delta \cdot \mathcal{C}_1 \quad \Gamma, \psi \rightarrow \Delta \cdot \mathcal{C}_2}{\Gamma, \varphi \vee \psi, \Xi \rightarrow \Delta \cdot (\mathcal{C}_1 \wedge \mathcal{C}_2)} (\vee \rightarrow) \\
\\
\frac{\Gamma \rightarrow \varphi \cdot \mathcal{C}}{\Gamma \rightarrow \varphi \vee \psi \cdot \mathcal{C}} (\rightarrow \vee_1) \qquad \frac{\Gamma \rightarrow \psi \cdot \mathcal{C}}{\Gamma \rightarrow \varphi \vee \psi \cdot \mathcal{C}} (\rightarrow \vee_2) \\
\\
\frac{\Gamma, \psi, \Xi \rightarrow \Delta \cdot \mathcal{C}_1 \quad \Gamma, \varphi \supset \psi, \Xi \rightarrow \varphi \cdot \mathcal{C}_2}{\Gamma, \varphi \supset \psi, \Xi \rightarrow \Delta \cdot (\mathcal{C}_1 \wedge \mathcal{C}_2)} (\supset \rightarrow) \\
\\
\frac{\varphi, \Gamma \rightarrow \psi \cdot \mathcal{C}}{\Gamma \rightarrow \varphi \supset \psi \cdot \mathcal{C}} (\rightarrow \supset) \\
\\
\frac{\Gamma \rightarrow \varphi \cdot \mathcal{C}}{\Gamma, \neg \varphi, \Xi \rightarrow \mathcal{C}} (\neg \rightarrow) \qquad \frac{\varphi, \Gamma \rightarrow \mathcal{C}}{\Gamma \rightarrow \neg \varphi \cdot \mathcal{C}} (\rightarrow \neg) \\
\\
\frac{\Gamma, \varphi[x \mapsto y], \forall x \varphi, \Xi \rightarrow \Delta \cdot \mathcal{C}}{\Gamma, \forall x \varphi, \Xi \rightarrow \Delta \cdot \exists y \mathcal{C}} (\forall \rightarrow) \\
\\
\frac{\Gamma \rightarrow \varphi[x \mapsto y] \cdot \mathcal{C}}{\Gamma \rightarrow \forall x \varphi \cdot (\exists y (y \not\in \{v_1, \dots, v_n\} \wedge \mathcal{C}))} (\rightarrow \forall) \\
\\
\frac{\Gamma, \varphi[x \mapsto y] \rightarrow \Delta \cdot \mathcal{C}}{\Gamma, \exists x \varphi \rightarrow \Delta \cdot (\exists y (y \not\in \{v_1, \dots, v_n\} \wedge \mathcal{C}))} (\exists \rightarrow) \\
\\
\frac{\Gamma \rightarrow \varphi[x \mapsto y] \cdot \mathcal{C}}{\Gamma \rightarrow \exists x \varphi \cdot \exists y \mathcal{C}} (\rightarrow \exists)
\end{array}$$

In the rule (Ax) , A is a predicate symbol different from equality. In the rules $(\forall \rightarrow)$ – $(\rightarrow \exists)$ the variable y has no free occurrences in the sequent in the conclusion of the rules. In the rule $(\rightarrow \forall)$, v_1, \dots, v_n are all free variables of $\Gamma, \forall x \varphi$. In the rule $(\exists \rightarrow)$, v_1, \dots, v_n are all free variables of $\Gamma, \exists x \varphi, \Delta$.

Figure 17: Calculus $\mathbf{LJ}_c^=$

example, the constraint is unsatisfiable. As in Figures 15 and 16, we denote by φ the formula $\forall x(x \approx a \vee x \approx b \supset \exists y(f(x, y) \approx b))$.

$$\begin{array}{c}
\frac{\varphi \rightarrow f(f(u, v), w) \approx b \vdash_{\forall} f(f(u, v), w) \approx b}{\varphi \rightarrow \exists w(f(f(u, v), w) \approx b) \cdot \exists w(\vdash_{\forall} f(f(u, v), w) \approx b)} (\rightarrow \exists) \\
\frac{\varphi \rightarrow \exists w \exists w(f(f(u, v), w) \approx b) \cdot \exists v \exists w(\vdash_{\forall} f(f(u, v), w) \approx b)}{\varphi \rightarrow \exists v \exists w \exists w(f(f(u, v), w) \approx b) \cdot \exists u \exists v \exists w(\vdash_{\forall} f(f(u, v), w) \approx b)} (\rightarrow \exists) \\
\frac{\varphi \rightarrow \exists u \exists v \exists w(f(f(u, v), w) \approx b) \cdot \exists u \exists v \exists w(\vdash_{\forall} f(f(u, v), w) \approx b)}{\rightarrow \varphi \supset \exists u \exists v \exists w(f(f(u, v), w) \approx b) \cdot \exists u \exists v \exists w(\vdash_{\forall} f(f(u, v), w) \approx b)} (\rightarrow \supset)
\end{array}$$

The resulting constraint $\exists u \exists v \exists w(\vdash_{\forall} f(f(u, v), w) \approx b)$ of this derivation is evidently unsatisfiable.

Another derivation of this formula has the same skeleton as that of Figures 15 and 16 and is shown in Figure 18.

Here the constraints C_1 – C_{12} are as follows:

$$\begin{array}{ll}
C_1 & \Rightarrow f(x_1, y_1) \approx b, f(x_2, y_2) \approx b \vdash_{\forall} f(f(u, v), w) \approx b \\
C_2 & \Rightarrow \exists w(f(x_1, y_1) \approx b, f(x_2, y_2) \approx b \vdash_{\forall} f(f(u, v), w) \approx b) \\
C_3 & \Rightarrow f(x_1, y_1) \approx b \vdash_{\forall} x_2 \approx b \\
C_4 & \Rightarrow \exists y_2(y_2 \not\approx \{x_1, y_1, x_2, u, v\} \wedge \\
& \quad \exists w(f(x_1, y_1) \approx b, f(x_2, y_2) \approx b \vdash_{\forall} f(f(u, v), w) \approx b)) \\
C_5 & \Rightarrow C_4 \wedge C_3 \\
C_6 & \Rightarrow \exists x_2(C_4 \wedge C_3) \\
C_7 & \Rightarrow \exists v \exists x_2(C_4 \wedge C_3) \\
C_8 & \Rightarrow \vdash_{\forall} x_1 \approx a \\
C_9 & \Rightarrow \exists y_1(y_1 \not\approx \{x_1, u\} \wedge \exists v \exists x_2(C_4 \wedge C_3)) \\
C_{10} & \Rightarrow \exists y_1(y_1 \not\approx \{x_1, u\} \wedge \exists v \exists x_2(C_4 \wedge C_3)) \wedge \vdash_{\forall} x_1 \approx a \\
C_{11} & \Rightarrow \exists x_1(\exists y_1(y_1 \not\approx \{x_1, u\} \wedge \exists v \exists x_2(C_4 \wedge C_3)) \wedge \vdash_{\forall} x_1 \approx a) \\
C_{12} & \Rightarrow \exists u \exists x_1(\exists y_1(y_1 \not\approx \{x_1, u\} \wedge \exists v \exists x_2(C_4 \wedge C_3)) \wedge \vdash_{\forall} x_1 \approx a)
\end{array}$$

The resulting constraint C_{12} of this derivation is satisfiable. To establish this, we first note that C_{12} is satisfiable if and only if such is the following constraint C obtained from C_{12} by removing all quantifiers:

$$\begin{array}{l}
f(x_1, y_1) \approx b, f(x_2, y_2) \approx b \vdash_{\forall} f(f(u, v), w) \approx b \wedge \\
f(x_1, y_1) \approx b \vdash_{\forall} x_2 \approx b \wedge \\
y_2 \not\approx \{x_1, y_1, x_2, u, v\} \wedge \\
\vdash_{\forall} x_1 \approx a \wedge \\
y_1 \not\approx \{x_1, u\}
\end{array}$$

This quantifier-free constraint is satisfied by the substitution $[u \mapsto a, x_1 \mapsto a, v \mapsto y_1, x_2 \mapsto b, w \mapsto y_2]$. Note that this substitutions was used for variable instantiation in γ -rules in the derivation of Figures 15 and 16.

$$\begin{array}{c}
\frac{\frac{f(x_1, y_1) \approx b, f(x_2, y_2) \approx b, \varphi \rightarrow f(f(u, v), w) \approx b \cdot C_1}{f(x_1, y_1) \approx b, f(x_2, y_2) \approx b, \varphi \rightarrow \exists w(f(f(u, v), w) \approx b) \cdot C_2} (\approx) (\rightarrow \exists)}{\frac{f(x_1, y_1) \approx b, \exists y(f(x_2, y) \approx b), \varphi \rightarrow \exists w(f(f(u, v), w) \approx b) \cdot C_4}{f(x_1, y_1) \approx b, x_2 \approx a \vee x_2 \approx b \supset \exists y(f(x_2, y) \approx b), \varphi \rightarrow \exists w(f(f(u, v), w) \approx b) \cdot C_5} (\exists \rightarrow)} \\
\frac{\frac{\frac{\dots \rightarrow x_2 \approx b \cdot C_3}{\dots \rightarrow x_2 \approx a \vee x_2 \approx b \cdot C_3} (\approx)}{\dots \rightarrow x_2 \approx a \vee x_2 \approx b \cdot C_5} (\rightarrow \vee_2)}{\dots \rightarrow x_2 \approx a \vee x_2 \approx b \cdot C_5} (\supset \rightarrow) \\
\frac{\frac{f(x_1, y_1) \approx b, \varphi \rightarrow \exists w(f(f(u, v), w) \approx b) \cdot C_6}{f(x_1, y_1) \approx b, \varphi \rightarrow \exists v \exists w(f(f(u, v), w) \approx b) \cdot C_7} (\rightarrow \exists)}{\frac{\exists y(f(x_1, y) \approx b), \varphi \rightarrow \exists v \exists w(f(f(u, v), w) \approx b) \cdot C_9}{x_1 \approx a \vee x_1 \approx b \supset \exists y(f(x_1, y) \approx b), \varphi \rightarrow \exists v \exists w(f(f(u, v), w) \approx b) \cdot C_{10}} (\exists \rightarrow)} \\
\frac{\frac{\frac{\frac{\dots \rightarrow x_1 \approx a \cdot C_8}{\dots \rightarrow x_1 \approx a \vee x_1 \approx b \cdot C_8} (\approx)}{\dots \rightarrow x_1 \approx a \vee x_1 \approx b \cdot C_{10}} (\approx)}{\dots \rightarrow x_1 \approx a \vee x_1 \approx b \cdot C_{10}} (\rightarrow \vee_1)}{\dots \rightarrow x_1 \approx a \vee x_1 \approx b \cdot C_{10}} (\supset \rightarrow) \\
\frac{\frac{\varphi \rightarrow \exists v \exists w(f(f(u, v), w) \approx b) \cdot C_{11}}{\varphi \rightarrow \exists u \exists v \exists w(f(f(u, v), w) \approx b) \cdot C_{12}} (\rightarrow \exists)}{\varphi \supset \exists u \exists v \exists w(f(f(u, v), w) \approx b) \cdot C_{12}} (\rightarrow \supset)
\end{array}$$

Figure 18: A derivation in \mathbf{LJ}_c^-

The correspondence of this derivation in \mathbf{LJ}_c^- to the derivation in \mathbf{LJ}^- is not coincidental: derivations in \mathbf{LJ}_c^- characterize all derivations in \mathbf{LJ}^- with a given skeleton in the following way:

9.4. THEOREM. *Let $\Gamma \rightarrow \varphi$ be a sequent, θ be a substitution and S be a skeleton. Then the following conditions are equivalent:*

1. *there is a derivation of $\Gamma\theta \rightarrow \varphi\theta$ in \mathbf{LJ}^- with the skeleton S ;*
2. *there is a constraint C and a derivation of $\Gamma \rightarrow \varphi \cdot C$ in \mathbf{LJ}_c^- with the skeleton S such that $\theta \models C$.*

This statement also gives us the completeness of \mathbf{LJ}_c^- :

9.5. THEOREM (completeness of \mathbf{LJ}_c^-). *Let S be a closed sequent. Then S is derivable in \mathbf{LJ}^- if and only if there is a satisfiable constraint C such that $S \cdot C$ is derivable in \mathbf{LJ}_c^- .*

Note that the set of constraints C such that $S \cdot C$ is derivable in \mathbf{LJ}_c^- with a skeleton S is *uniquely defined by the skeleton S* up to renaming of bound variables. Thus, the proof-search in intuitionistic logic can be considered as consisting of two parts: constructing a skeleton and the constraint satisfaction. This is similar to the matrix characterization of provability for classical logic.

The skeleton instantiation problem is undecidable which causes doubts about the efficiency of such procedures. However, for intuitionistic logic, because of simple reduction of simultaneous rigid E -unifiability to the derivability problem, handling of simultaneous rigid E -unification is inevitable for designing a complete procedure. In addition, some special cases of simultaneous rigid E -unification are decidable [Degtyarev, Matiyasevich and Voronkov 1995, Degtyarev, Matiyasevich and Voronkov 1996] (for example, the function-free case is NP-complete) which allows us to use decision procedures for these fragments.

9.4. Modal logics with equality

There is no uniform viewpoint on the use of quantification in modal logics. Neither is there a uniform viewpoint on the use of equality. The survey of Garson [1984] contains some material on this topic. Without going into many details, we show that the above considerations about intuitionistic logic can as well be applied to some formalizations of various modal logics. Readers not familiar with modal logics can skip this section. We consider modal logics for which equality is axiomatized by the rules (*refl*) and (\approx), like for \mathbf{LK}^- or \mathbf{LJ}^- . As an example, we take $\mathbf{S4}^-$.

We will use the following notation. For any multiset of formulas Γ ,

- ▶ $\Box\Gamma$ (respectively, $\Diamond\Gamma$) denotes the multiset of formulas $\{\Box\varphi \mid \varphi \in \Gamma\}$ (respectively, $\{\Diamond\varphi \mid \varphi \in \Gamma\}$).
- ▶ The *sequent calculus* $\mathbf{S4}^-$ for modal logic with equality is obtained from \mathbf{LK}^- by adding the following inference rules:

$$\begin{array}{c}
\frac{\Gamma, T \varphi}{\Gamma, T \Box \varphi} (T \Box) \qquad \frac{T \Gamma, F \Delta, F \varphi}{\Xi, T \Box \Gamma, F \Diamond \Delta, F \Box \varphi} (F \Box) \\
\\
\frac{T \Gamma, F \Delta, T \varphi}{\Xi, T \Box \Gamma, F \Diamond \Delta, T \Diamond \varphi} (T \Diamond) \qquad \frac{\Gamma, F \varphi}{\Gamma, F \Diamond \varphi} (F \Diamond)
\end{array}$$

Consider again any system R of rigid equations of the form (9.1) and the formula ψ_R of the form (9.2) (see page 683). Let $\varphi \doteq \exists x_1 \dots \exists x_m \Box \psi_R$. Note that φ does not contain \Diamond , and therefore for any derivation using the rule $(F \Box)$, the premise of this rule will consist of one formula $F \psi_R \sigma$, for some substitution σ . Therefore, any derivation of φ in $S4^=$ has the following form:

$$\left. \begin{array}{c}
\vdots \\
F \psi_R \sigma
\end{array} \right\} \text{derivation of } \psi_R \sigma$$

$$\left. \begin{array}{c}
\frac{\Xi, F \Box \psi_R \sigma}{\Xi, F \Box \psi_R \sigma} (\nu) \\
F \exists x_1 \dots \exists x_m \Box \psi_R
\end{array} \right\} \text{derivation using only } \gamma\text{-rules}$$

Again, such a derivation exists if and only if σ is a solution to R . Thus, nearly every result asserted above for $LJ^=$, can be reformulated in a suitable way for $S4^=$.

The necessity of handling simultaneous rigid E -unification will arise in almost any logic whose semantics is based on possible worlds. The same trick that we used for modal logics above can, for instance, be used for some temporal logics.

10. Conclusion and open problems

The first version of this chapter was written in 1996. For the three years after 1996 most results in the area of equality reasoning in sequent calculi were obtained in the area simultaneous rigid E -unification and its connections to second-order unification [Veanes 1998b]. In particular, new results on undecidable fragments of second-order unification were obtained in [Veanes 1998a, Veanes 1998b]. Ganzinger, Jacquemard and Veanes [1998] introduced a new notion of *rigid reachability*, which is essentially an oriented version of simultaneous rigid E -unification. Some results on rigid reachability were proved in [Cortier, Ganzinger, Jacquemard and Veanes 1999]. However, there have been essentially no new results related to *automated reasoning procedures* for sequent calculi with equality.

In this section we will briefly mention possible research directions and open problems.

10.1. Simultaneous rigid *E*-unification

One open problem is the decidability of *monadic simultaneous rigid E-unification*, that is simultaneous rigid *E*-unification in the signatures with unary function symbols only. [Gurevich and Voronkov 1997a, Gurevich and Voronkov 1997b, Gurevich and Voronkov 1999] prove the equivalence of this problem to an extension of word equations.

Other relevant open problems (for example, decidability and complexity of various formula instantiation problems) are mentioned in [Voronkov 1998a, Voronkov 1999, Degtyarev, Gurevich and Voronkov 2000].

10.2. Equality reasoning for model elimination and other tableau-based methods

In the area of tableau-based methods, and especially in model-elimination based procedures, two variants of equality elimination were proposed in [Degtyarev and Voronkov 1996b, Moser et al. 1995]. However, further research is required here. One major problem is a proof of completeness of equality elimination with various redundancies, both in the equality elimination part (i.e., generation of solution clauses) and in the tableau part of the derivation. Theorem 8.3 asserts that the solution clauses are sufficient, but it does not assert that the generation of solution clauses can be further constrained by redundancy criteria, for example, simplification by unit equalities. In order to prove completeness in presence of a collection of redundancy criteria *C* for the generation of solution clauses, it is enough to prove the following statement:

Let S be a set of clauses with the following property: the arguments of every non-equality atom in S are variables. Then there exists a refutation of S with redundancy criteria C in which applications of superposition precede applications of other rules (resolution, equality resolution, and factoring).

It is also interesting to prove completeness in presence of simplification rules in the tableau part of the proof (for example, simplification by the unit equalities generated in the equality elimination part).

An interesting problem is the extension of the basic folding method proposed in [Degtyarev and Voronkov 1996c, Degtyarev and Voronkov 2001a] for Horn clauses to more general sets of formulas.

Another very interesting problem is the actual implementation of equality elimination and basic folding in a way competitive with resolution-based methods.

10.3. Equality reasoning in nonclassical logics

There are formalizations of equality for nonclassical logics different from those of Section 9 [see e.g. Garson 1984, Fitting 1991]. Automated reasoning methods for such formalizations should be developed.

An interesting problem is to find nonclassical logics with equality in which equality reasoning can be done in the same way as in classical logics, so that paramodulation-based methods can be used instead of simultaneous rigid *E*-unification.

Acknowledgments

Both authors were partially supported by TFR grants. Anatoli was also supported by grants of the Swedish Institute and Swedish Royal Academy of Sciences.

Bibliography

- ANDREWS P. [1976], 'Refutations by matings', *IEEE Trans. Comput.* **25**, 801–807.
- ANDREWS P. [1981], 'Theorem proving via general matings', *Journal of the Association for Computing Machinery* **28**(2), 193–214.
- ANDREWS P. [1986], *An introduction to type theory: to truth through proof*, Academic Press.
- AVRON A. [1993], 'Gentzen-type systems, resolution and tableaux', *Journal of Automated Reasoning* **10**, 256–281.
- BAAZ M. [1993], Note on the existence of most general semi-unifiers, in 'Arithmetic, Proof Theory and Computation Complexity', Vol. 23 of *Oxford Logic Guides*, Oxford University Press, pp. 20–29.
- BAAZ M. AND FERMÜLLER C. [1995], Non-elementary speedups between different versions of tableaux, in P. Baumgartner, R. Hähnle and J. Posegga, eds, 'Theorem Proving with Analytic Tableaux and Related Methods', number 918 in 'Lecture Notes in Artificial Intelligence', Schloß Rheinfels, St. Goar, Germany, pp. 217–230.
- BAAZ M. AND LEITSCH A. [1994], 'On skolemization and proof complexity', *Fundamenta Informaticae* **20**(4).
- BACHMAIR L. AND GANZINGER H. [1990], On restriction of ordered paramodulation with simplification, in M. Stickel, ed., 'Proc. 10th CADE', Vol. 449 of *Lecture Notes in Artificial Intelligence*, pp. 427–441.
- BACHMAIR L. AND GANZINGER H. [1991], Completion of first-order clauses with equality by strict superposition, in 'Conditional and Typed Rewriting Systems', Vol. 516 of *Lecture Notes in Computer Science*, Springer Verlag, Montreal, pp. 164–180.
- BACHMAIR L. AND GANZINGER H. [1994], 'Rewrite-based equational theorem proving with selection and simplification', *Journal of Logic and Computation* **4**(3), 217–247.
- BACHMAIR L. AND GANZINGER H. [1998], Strict basic superposition, in C. Kirchner and H. Kirchner, eds, 'Automated Deduction — CADE-15. 15th International Conference on Automated Deduction', Vol. 1421 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Lindau, Germany, pp. 160–174.
- BACHMAIR L., GANZINGER H., LYNCH C. AND SNYDER W. [1992], Basic paramodulation and superposition, in D. Kapur, ed., '11th International Conference on Automated Deduction', Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Saratoga Springs, NY, USA, pp. 462–476.
- BACHMAIR L., GANZINGER H., LYNCH C. AND SNYDER W. [1995], 'Basic paramodulation', *Information and Computation* **121**, 172–192.
- BACHMAIR L., GANZINGER H. AND VORONKOV A. [1998], Elimination of equality via transformation with ordering constraints, in C. Kirchner and H. Kirchner, eds, 'Automated Deduction — CADE-15. 15th International Conference on Automated Deduction', Vol. 1421 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Lindau, Germany, pp. 175–190.

- BAUMGARTNER P. AND FURBACH U. [1993], 'Consolution as a framework for comparing calculi', *Journal of Symbolic Computations* **16**, 445-477.
- BAUMGARTNER P. AND FURBACH U. [1994], PROTEIN: A PROver with a Theory Extension Interface, in A. Bundy, ed., 'Automated Deduction — CADE-12. 12th International Conference on Automated Deduction.', Vol. 814 of *Lecture Notes in Artificial Intelligence*, Nancy, France, pp. 769-773.
- BECHER G. AND PETERMANN U. [1994], Rigid unification by completion and rigid paramodulation, in B. Nebel and L. Dreschler-Fischer, eds, 'KI-94: Advances in Artificial Intelligence. 18th German Annual Conference on Artificial Intelligence', Vol. 861 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Saarbrücken, Germany, pp. 319-330.
- BECKERT B. [1994], A completion-based method for mixed universal and rigid *E*-unification, in A. Bundy, ed., 'Automated Deduction — CADE-12. 12th International Conference on Automated Deduction.', Vol. 814 of *Lecture Notes in Artificial Intelligence*, Nancy, France, pp. 678-692.
- BECKERT B. [1995], Are minimal solutions to simultaneous rigid *E*-unification sufficient for adding equality to semantic tableaux?, Privately circulated manuscript, University of Karlsruhe.
- BECKERT B. [1997a], Equality and other theory inferences, in M. D'Agostino, D. Gabbay, R. Hähnle and J. Posegga, eds, 'Handbook of Tableau Methods', Kluwer.
- BECKERT B. [1997b], 'Semantic tableaux with equality', *Journal of Logic and Computation* **7**(1), 38-58.
- BECKERT B. AND HÄHNLE R. [1992], An improved method for adding equality to free variable semantic tableaux, in D. Kapur, ed., '11th International Conference on Automated Deduction (CADE)', Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Saratoga Springs, NY, USA, pp. 678-692.
- BECKERT B., HÄHNLE R. AND SCHMITT P. [1993], The even more liberalized δ -rule in free variable semantic tableaux, in G. Gottlob, A. Leitsch and D. Mundici, eds, 'Computational Logic and Proof Theory. Proceedings of the Third Kurt Gödel Colloquium, KGC'93', Vol. 713 of *Lecture Notes in Computer Science*, Brno, pp. 108-119.
- BELTYUKOV A. [1976], 'Decidability of the universal theory of natural numbers with addition and divisibility (in Russian)', *Zapiski Nauchnyh Seminarov LOMI* **60**, 15-28. English translation in: *Journal of Soviet Mathematics*.
- BENANAV D. [1990], Simultaneous paramodulation, in M. Stickel, ed., 'Proc. 10th Int. Conf. on Automated Deduction', Vol. 449 of *Lecture Notes in Artificial Intelligence*, pp. 442-455.
- BETH E. [1959], *The Foundations of Mathematics*, North Holland.
- BETH E. [1983], On machines which prove theorems, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 79-92. Originally appeared in 1958.
- BIBEL W. [1981], 'On matrices with connections', *Journal of the Association for Computing Machinery* **28**(4), 633-645.
- BIBEL W. [1993], *Deduction. Automated Logic.*, Academic Press.
- BIBEL W., BRÜNING S., EGLY U., KORN D. AND RATH T. [1995], Issues in theorem proving based on the connection method, in P. Baumgartner, R. Hähnle and J. Posegga, eds, 'Theorem Proving with Analytic Tableaux and Related Methods', number 918 in 'Lecture Notes in Artificial Intelligence', Schloß Rheinfels, St. Goar, Germany, pp. 1-16.
- BIBEL W. AND SCHREIBER J. [1975], Proof search in a Gentzen-like system of first order logic, in E. Gelenbe and D. Potier, eds, 'Proc. Int. Computing Symp.', North Holland, pp. 205-212.
- BOY DE LA TOUR T. [1990], Minimizing the number of clauses by renaming, in M. Stickel, ed., 'Proc. 10th CADE', Vol. 449 of *Lecture Notes in Artificial Intelligence*, pp. 558-572.
- BRAND D. [1975], 'Proving theorems with the modification method', *SIAM Journal of Computing* **4**, 412-430.

- CHANG C. [1972], 'Theorem proving with variable-constrained resolution', *Information Sciences* 4, 217-231.
- CORTIER V., GANZINGER H., JACQUEMARD F. AND VEANES M. [1999], Decidable fragments of simultaneous rigid reachability, in J. Wiedermann, P. van Emde Boas and M. Nielsen, eds, 'Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings', Vol. 1644 of *Lecture Notes in Computer Science*, pp. 250-260.
- CURRY H. [1963], *Foundations of Mathematical Logic*, New York.
- DARLINGTON J. [1968], Automatic theorem proving with equality substitutions and mathematical induction, in B. Meltzer and D. Michie, eds, 'Machine Intelligence', Vol. 3, American Elsevier, N.Y., pp. 113-127.
- DE KOGEL E. [1995], Rigid E -unification simplified, in P. Baumgartner, R. Hähnle and J. Posegga, eds, 'Theorem Proving with Analytic Tableaux and Related Methods', number 918 in 'Lecture Notes in Artificial Intelligence', Schloß Rheinfels, St. Goar, Germany, pp. 17-30.
- DEGTYAREV A. [1979], The strategy of monotone paramodulation (in Russian), in 'Fifth Soviet All-Union Conference on Mathematical Logic', Novosibirsk, p. 39.
- DEGTYAREV A. [1982], On the forms of inference in calculi with equality and paramodulation, in Y. Kapitonova, ed., 'Automation of Research in Mathematics', Institute of Cybernetics, Kiev, Kiev, pp. 14-26.
- DEGTYAREV A. [1985], Rewriting strategies for computational logic (in Russian), in 'Proc. of the Soviet Conf. on Applied Logic', Novosibirsk, pp. 69-72.
- DEGTYAREV A., GUREVICH Y., NARENDRAN P., VEANES M. AND VORONKOV A. [1997], The decidability of simultaneous rigid E -unification with one variable, UPMail Technical Report 139, Uppsala University, Computing Science Department.
- DEGTYAREV A., GUREVICH Y., NARENDRAN P., VEANES M. AND VORONKOV A. [1998], The decidability of simultaneous rigid E -unification with one variable, in T. Nipkow, ed., 'Rewriting Techniques and Applications, RTA'98', Vol. 1379 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 181-195.
- DEGTYAREV A., GUREVICH Y., NARENDRAN P., VEANES M. AND VORONKOV A. [2000], 'The decidability of simultaneous rigid E -unification with one variable and related results', *Theoretical Computer Science* 243(1-2), 167-184.
- DEGTYAREV A., GUREVICH Y. AND VORONKOV A. [1996], Herbrand's theorem and equational reasoning: Problems and solutions, in 'Bulletin of the European Association for Theoretical Computer Science', Vol. 60, pp. 78-95. The "Logic in Computer Science" column.
- DEGTYAREV A., GUREVICH Y. AND VORONKOV A. [2000], Herbrand's theorem and equational reasoning: Problems and solutions, in Y. Gurevich, ed., 'Logic in Computer Science', World Scientific. to appear.
- DEGTYAREV A., KOVAL Y. AND VORONKOV A. [1995], Handling equality in logic programming via basic folding, UPMail Technical Report 101, Uppsala University, Computing Science Department. Revised June 11, 1997.
- DEGTYAREV A., MATIYASEVICH Y. AND VORONKOV A. [1995], Simultaneous rigid E -unification is not so simple, UPMail Technical Report 104, Uppsala University, Computing Science Department.
- DEGTYAREV A., MATIYASEVICH Y. AND VORONKOV A. [1996], Simultaneous rigid E -unification and related algorithmic problems, in 'Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS'96)', IEEE Computer Society Press, New Brunswick, NJ, pp. 494-502.
- DEGTYAREV A. AND VORONKOV A. [1986], 'Equality control methods in machine theorem proving', *Cybernetics* 22(3), 298-307.
- DEGTYAREV A. AND VORONKOV A. [1994a], Equality elimination for semantic tableaux, UPMail Technical Report 90, Uppsala University, Computing Science Department.
- DEGTYAREV A. AND VORONKOV A. [1994b], A new procedural interpretation of Horn clauses with equality, UPMail Technical Report 89, Uppsala University, Computing Science Department.

- DEGTYAREV A. AND VORONKOV A. [1995a], Equality elimination for the inverse method and extension procedures, in C. Mellish, ed., 'Proc. International Joint Conference on Artificial Intelligence (IJCAI)', Vol. 1, Montréal, pp. 342–347.
- DEGTYAREV A. AND VORONKOV A. [1995b], General connections via equality elimination, in M. De Glas and Z. Pawlak, eds, 'Second World Conference on the Fundamentals of Artificial Intelligence (WOCFAI-95)', Angkor, Paris, pp. 109–120.
- DEGTYAREV A. AND VORONKOV A. [1995c], A new procedural interpretation of Horn clauses with equality, in L. Sterling, ed., 'Proceedings of the Twelfth International Conference on Logic Programming', The MIT Press, pp. 565–579.
- DEGTYAREV A. AND VORONKOV A. [1995d], Reduction of second-order unification to simultaneous rigid E -unification, UPMail Technical Report 109, Uppsala University, Computing Science Department.
- DEGTYAREV A. AND VORONKOV A. [1995e], Simultaneous rigid E -unification is undecidable, UPMail Technical Report 105, Uppsala University, Computing Science Department.
- DEGTYAREV A. AND VORONKOV A. [1996a], Decidability problems for the prenex fragment of intuitionistic logic, in 'Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS'96)', IEEE Computer Society Press, New Brunswick, NJ, pp. 503–512.
- DEGTYAREV A. AND VORONKOV A. [1996b], Equality elimination for the tableau method, in J. Calmet and C. Limongelli, eds, 'Design and Implementation of Symbolic Computation Systems. International Symposium, DISCO'96', Vol. 1128 of *Lecture Notes in Computer Science*, Karlsruhe, Germany, pp. 46–60.
- DEGTYAREV A. AND VORONKOV A. [1996c], Handling equality in logic programs via basic folding, in R. Dychhoff, H. Herre and P. Schroeder-Heister, eds, 'Extensions of Logic Programming (5th International Workshop, ELP'96)', Vol. 1050 of *Lecture Notes in Computer Science*, Leipzig, Germany, pp. 119–136.
- DEGTYAREV A. AND VORONKOV A. [1996d], 'A note on semantics of logics programs with equality based on complete sets of E -unifiers', *Journal of Logic Programming* 28(3), 207–216.
- DEGTYAREV A. AND VORONKOV A. [1996e], Simultaneous rigid E -unification is undecidable, in H. K. Büning, ed., 'Computer Science Logic. 9th International Workshop, CSL'95', Vol. 1092 of *Lecture Notes in Computer Science*, Paderborn, Germany, September 1995, pp. 178–190.
- DEGTYAREV A. AND VORONKOV A. [1996f], 'The undecidability of simultaneous rigid E -unification', *Theoretical Computer Science* 166(1–2), 291–300.
- DEGTYAREV A. AND VORONKOV A. [1996g], What you always wanted to know about rigid E -unification, in J. Alferes, L. Pereira and E. Orłowska, eds, 'Logics in Artificial Intelligence. European Workshop, JELIA'96', Vol. 1126 of *Lecture Notes in Artificial Intelligence*, Evora, Portugal, pp. 50–69.
- DEGTYAREV A. AND VORONKOV A. [1997], What you always wanted to know about rigid E -unification, UPMail Technical Report 143, Uppsala University, Computing Science Department.
- DEGTYAREV A. AND VORONKOV A. [1998], 'What you always wanted to know about rigid E -unification', *Journal of Automated Reasoning* 20(1), 47–80.
- DEGTYAREV A. AND VORONKOV A. [2001a], 'Handling equality in logic programming via basic folding', *Journal of Symbolic Computations* pp. 1–37. to appear.
- DEGTYAREV A. AND VORONKOV A. [2001b], The inverse method, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 4, pp. 179–272.
- DEGTYAREV A. AND VORONKOV A. [2001c], Kanger's choices in automated reasoning, in G. Holmström-Hintikka, S. Lindström and R. Sliwiski, eds, 'Collected papers of Stig Kanger with Essays on his Life and Work', Vol. II, Kluwer Academic Publishers, pp. 1–15.
- DOWEK G. [2001], Higher-order unification and matching, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 16, pp. 1009–1062.

- EDER E. [1984], An implementation of a theorem prover based on the connection method, in W. Bibel and B. Petkoff, eds, 'AIMSA'84, Artificial Intelligence — Methodology, Systems, Application', North Holland, pp. 121–128.
- EDER E. [1988], A comparison of the resolution calculus and the connection method, and a new calculus generalizing both methods, in E. Börger, G. Jäger, H. Kleine Büning and M. Richter, eds, 'CSL'88 (Proc. 2nd Workshop on Computer Science Logic)', Vol. 385 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 80–98.
- EDER E. [1991], Consolution and its relation with resolution, in 'Proc. International Joint Conference on Artificial Intelligence (IJCAI)', pp. 132–136.
- FITTING M. [1990], *First Order Logic and Automated Theorem Proving*, Springer Verlag, New York.
- FITTING M. [1991], Modal logic should say more than it does, in J.-L. Lassez and G. Plotkin, eds, 'Computational Logic. Essays in Honor of Alan Robinson.', The MIT Press, Cambridge, MA, pp. 113–135.
- FITTING M. [1994], 'Tableaux for logic programming', *Journal of Automated Reasoning* **13**, 175–188.
- FITTING M. [1996], *First Order Logic and Automated Theorem Proving*, Springer Verlag, New York. Second edition.
- GALLIER J. [1986], *Logic for Computer Science: Foundations of Automatic Theorem Proving*, Harper and Row, New York.
- GALLIER J. [1992], Unification procedures in automated deduction methods based on matings: a survey, in M. Nivat and A. Podelski, eds, 'Tree Automata and Languages', Elsevier Science, pp. 439–485.
- GALLIER J., NARENDRA P., PLAISTED D. AND SNYDER W. [1988], Rigid *E*-unification is NP-complete, in 'Proc. IEEE Conference on Logic in Computer Science (LICS)', IEEE Computer Society Press, pp. 338–346.
- GALLIER J., NARENDRA P., PLAISTED D. AND SNYDER W. [1990], 'Rigid *E*-unification: NP-completeness and applications to equational matings', *Information and Computation* **87**(1/2), 129–195.
- GALLIER J., NARENDRA P., RAATZ S. AND SNYDER W. [1992], 'Theorem proving using equational matings and rigid *E*-unification', *Journal of the Association for Computing Machinery* **39**(2), 377–429.
- GALLIER J., RAATZ S. AND SNYDER W. [1987], Theorem proving using rigid *E*-unification: Equational matings, in 'Proc. IEEE Conference on Logic in Computer Science (LICS)', IEEE Computer Society Press, pp. 338–346.
- GALLIER J., RAATZ S. AND SNYDER W. [1989], Rigid *E*-unification and its applications to equational matings, in H. Aït Kaci and M. Nivat, eds, 'Resolution of Equations in Algebraic Structures', Vol. 1, Academic Press, pp. 151–216.
- GALLIER J. AND SNYDER W. [1989], 'Complete sets of transformations for general *E*-unification', *Theoretical Computer Science* **67**, 203–260.
- GANZINGER G., JACQUEMARD F. AND VEANES M. [1998], Rigid reachability, in J. Hsiang and A. Ohori, eds, 'Advances in Computing Science - ASIAN'98, 4th Asian Computing Science Conference', Vol. 1538 of *Lecture Notes in Computer Science*, Springer Verlag, Manila, The Philippines, pp. 4–21.
- GARSON J. [1984], Quantification in modal logic, in D. Gabbay and F. Guenther, eds, 'Handbook in Philosophical Logic', Vol. II, D. Reidel Publishing Company, chapter II.5, pp. 249–307.
- GENTZEN G. [1934], 'Untersuchungen über das logische Schließen', *Mathematische Zeitschrift* **39**, 176–210, 405–431.
- GENTZEN G. [1936], 'Die Widerspruchsfreiheit der reinen Zahlentheorie', *Mathematische Annalen* **112**, 493–565.
- GIRARD J.-Y. [1987a], 'Linear logic', *Theoretical Computer Science* **50**(1), 1–101.

- GIRARD J.-Y. [1987b], *Proof Theory and Logical Complexity*, Studies in Proof Theory, Bibliopolis, Napoly.
- GOLDFARB W. [1981], 'The undecidability of the second-order unification problem', *Theoretical Computer Science* **13**, 225–230.
- GOUBAULT J. [1993], A rule-based algorithm for rigid E -unification, in G. Gottlob, A. Leitsch and D. Mundici, eds, 'Computational Logic and Proof Theory. Proceedings of the Third Kurt Gödel Colloquium, KGC'93', Vol. 713 of *Lecture Notes in Computer Science*, Brno, pp. 202–210.
- GOUBAULT J. [1994], Rigid \bar{E} -unifiability is DEXPTIME-complete, in 'Proc. IEEE Conference on Logic in Computer Science (LICS)', IEEE Computer Society Press.
- GUREVICH Y. AND VORONKOV A. [1997a], Monadic simultaneous rigid E -unification and related problems, UPMail Technical Report 137, Uppsala University, Computing Science Department. Revised June 20, 1997.
- GUREVICH Y. AND VORONKOV A. [1997b], Monadic simultaneous rigid E -unification and related problems, in P. Degano, R. Corrieri and A. Marchetti-Spaccamella, eds, 'Automata, Languages and Programming. 24th International Colloquium, ICALP'97', Vol. 1256 of *Lecture Notes in Computer Science*, Bologna, Italy, pp. 154–165.
- GUREVICH Y. AND VORONKOV A. [1999], 'Monadic simultaneous rigid E -unification', *Theoretical Computer Science* **222**, 133–152.
- HÄHNLE R. AND SCHMITT P. [1994], 'The liberalized δ -rule in free variable semantic tableaux', *Journal of Automated Reasoning* **13**, 211–221.
- HSIANG J. AND RUSINOWITCH M. [1986], A new method for establishing refutational completeness in theorem proving, in 'Proc. 8th CADE', Vol. 230 of *Lecture Notes in Computer Science*, pp. 141–152.
- HSIANG J. AND RUSINOWITCH M. [1991], 'Proving refutational completeness of theorem proving strategies: the transfinite semantic tree method', *Journal of the Association for Computing Machinery* **38**(3), 559–587.
- HULLOT J. [1980], Canonical forms and unification, in '5th CADE', Vol. 87 of *Lecture Notes in Computer Science*, pp. 318–334.
- IBENS O. AND LETZ R. [1997], Subgoal alternation in model elimination, in D. Galmiche, ed., 'Automated Reasoning with Analytic Tableaux and Related Methods', Vol. 1227 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, pp. 201–215.
- KANGER S. [1957], *Provability in Logic*, Vol. 1 of *Studies in Philosophy*, Almqvist and Wicksell, Stockholm.
- KANGER S. [1963], A simplified proof method for elementary logic, in P. Braffort and D. Hirschberg, eds, 'Computer Programming and Formal Systems', North Holland, pp. 87–94. Reprinted as [Kanger 1983].
- KANGER S. [1983], A simplified proof method for elementary logic, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 364–371. Originally published as [Kanger 1983].
- KLEENE S. [1952], *Introduction to Metamathematics*, Van Nostrand P.C., Amsterdam.
- KNUTH D. AND BENDIX P. [1970], Simple word problems in universal algebras, in J. Leech, ed., 'Computational Problems in Abstract Algebra', Pergamon Press, Oxford, pp. 263–297.
- LANKFORD D. [1975], Canonical inference, Technical report, Department of Mathematics, South-Western University, Georgetown, Texas.
- LEE R. AND CHANG C. [1973], *Symbolic Logic and Mechanical Theorem Proving*, Academic Press.
- LIFSCHITZ V. [1967], 'A normal form of derivations in predicate calculus with equality and function symbols (in Russian)', *Zapiski Nauchnyh Seminarov LOMI* **4**, 58–64. English Translation in: Seminars in Mathematics: Steklov Math. Inst. 4, Consultants Bureau, NY-London, 1969.
- LIFSCHITZ V. [1968], Specialized forms of derivation in predicate calculus with equality and functional symbols (in Russian), in 'Trudy MIAN', Vol. 98, pp. 5–25. English translation in: Proc. Steklov Institute of Math., AMS, Providence, RI, 1971.

- LIFSCHITZ V. [1989], 'What is the inverse method?', *Journal of Automated Reasoning* **5**(1), 1–23.
- LIPSHITZ L. [1978], 'The Diophantine problem for addition and divisibility', *Transactions of the American Mathematical Society* **235**, 271–283.
- LIPSHITZ L. [1981], 'Some remarks on the Diophantine problem for addition and divisibility', *Bull. Soc. Math. Belg. Sér. B* **33**(1), 41–52.
- LOVELAND D. [1978], *Automated Theorem Proving: a Logical Basis*, North Holland.
- LUSK E. [1992], Controlling redundancy in large search spaces: Argonne-style theorem proving through the years, in A. Voronkov, ed., 'Logic Programming and Automated Reasoning. International Conference LPAR'92.', Vol. 624 of *Lecture Notes in Artificial Intelligence*, St. Petersburg, Russia, pp. 96–106.
- LYNCH C. [1997], 'Oriented equational logic is complete', *Journal of Symbolic Computations* **23**(1), 23–45.
- MAKANIN G. [1977], 'The problem of solvability of equations in free semigroups', *Mat. Sbornik (in Russian)* **103**(2), 147–236. English Translation in American Mathematical Soc. Translations (2), vol. 117, 1981.
- MART'JANOV V. [1977], 'Universal extended theories of integers', *Algebra i Logika* **16**(5), 588–602.
- MASLOV S. [1964], 'The inverse method of establishing deducibility in the classical predicate calculus', *Soviet Mathematical Doklady* **5**, 1420–1424.
- MASLOV S. [1971], 'The generalization of the inverse method to predicate calculus with equality (in Russian)', *Zapiski Nauchnykh Seminarov LOMI* **20**, 80–96. English translation in: *Journal of Soviet Mathematics* 1, no. 1.
- MASLOV S. [1983a], An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning (Classical papers on Computational Logic)', Vol. 2, Springer Verlag, pp. 48–54.
- MASLOV S. [1983b], Relationship between tactics of the inverse method and the resolution method, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning (Classical papers on Computational Logic)', Vol. 2, Springer Verlag, pp. 264–272.
- MASLOV S. [1987], *Theory of Deductive Systems and its Applications*, MIT Press.
- MASLOV S. AND MINTS G. [1983], The proof-search theory and the inverse method (in Russian), in M. G., ed., 'Mathematical Logic and Automatic Theorem Proving', Nauka, Moscow, pp. 291–314.
- MASLOV S., MINTS G. AND OREVKOV V. [1983], Mechanical proof-search and the theory of logical deduction in the USSR, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning (Classical papers on Computational Logic)', Vol. 1, Springer Verlag, pp. 29–38.
- MATULIS V. [1962], 'Two variants of classical predicate calculus without structure rules (in Russian)', *Soviet Mathematical Doklady* **147**(5), 1029–1031.
- MATULIS V. [1963], 'On variants of classical predicate calculus with the unique deduction tree (in Russian)', *Soviet Mathematical Doklady* **148**, 768–770.
- MINKER J., RAJASEKAR A. AND LOBO J. [1991], Theory of disjunctive logic programs, in J.-L. Lassez and G. Plotkin, eds, 'Computational Logic. Essays in Honor of Alan Robinson.', The MIT Press, Cambridge, MA, pp. 613–639.
- MINTS G. [1990], Gentzen-type systems and resolution rules. Part I. Propositional logic, in P. Martin-Löf and G. Mints, eds, 'COLOG-88', Vol. 417 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 198–231.
- MINTS G., OREVKOV V. AND TAMMET T. [1996], Transfer of sequent calculus strategies to resolution, in 'Proof Theory of Modal Logic', *Studies in Pure and Applied Logic*, Kluwer Academic Publishers. To appear.
- MOSER M., LYNCH C. AND STEINBACH J. [1995], Model elimination with basic ordered paramodulation, Technical Report AR-95-11, Fakultät für Informatik, Technische Universität München, München.
- MOSER M. AND STEINBACH J. [1997], STE-modification revisited, Technical Report AR-97-03, Fakultät für Informatik, Technische Universität München, München.

- NIEUWENHUIS R. [1993], 'Simple LPO constraint solving methods', *Information Processing Letters* **47**, 65–69.
- NIEUWENHUIS R. AND RUBIO A. [1992a], Basic superposition is complete, in 'ESOP'92', Vol. 582 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 371–389.
- NIEUWENHUIS R. AND RUBIO A. [1992b], Theorem proving with ordering constrained clauses, in D. Kapur, ed., '11th International Conference on Automated Deduction (CADE)', Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Saratoga Springs, NY, USA, pp. 477–491.
- NIEUWENHUIS R. AND RUBIO A. [1995], 'Theorem proving with ordering and equality constrained clauses', *Journal of Symbolic Computations* **19**, 321–351.
- NONNENGART A. AND WEIDENBACH C. [2001], Computing small clause normal forms, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 6, pp. 335–367.
- NORGELA S. [1974], On the size of derivations under minus-normalization in Russian, in V. Smirnov, ed., 'The Theory of Logical Inference', Institute of Philosophy, Moscow.
- OREVKOV V. [1969], 'On nonlengthening applications of equality rules (in Russian)', *Zapiski Nauchnyh Seminarov LOMI* **16**, 152–156. English Translation in: *Seminars in Mathematics: Steklov Math. Inst. 16*, Consultants Bureau, NY-London, 1971, p.77–79.
- PAIS J. AND PETERSON G. [1991], 'Using forcing to prove completeness of resolution and paramodulation', *Journal of Symbolic Computations* **11**, 3–19.
- PETERMANN U. [1994], A complete connection calculus with rigid *E*-unification, in 'JELIA'94', Vol. 838 of *Lecture Notes in Computer Science*, pp. 152–166.
- PETERSON G. [1983], 'A technique for establishing completeness results in theorem proving with equality', *SIAM Journal of Computing* **12**(1), 82–100.
- PLAISTED D. [1995], Special cases and substitutes for rigid *E*-unification, Technical Report MPI-I-95-2-010, Max-Planck-Institut für Informatik.
- PLAISTED D. AND GREENBAUM S. [1986], 'A structure-preserving clause form translation', *Journal of Symbolic Computations* **2**, 293–304.
- PLIUŠKEVIČIENE A. [1969], 'Elimination of cut-type rules in axiomatic theories with equality (in Russian)', *Zapiski Nauchnyh Seminarov LOMI* **16**, 175–184. English Translation in: *Seminars in Mathematics: Steklov Math. Inst. 16*, Consultants Bureau, NY-London, 1971, p.90–94.
- PRAWITZ D. [1960], 'An improved proof procedure', *Theoria* **26**(2), 102–128. Reprinted as [Prawitz 1983].
- PRAWITZ D. [1983], An improved proof procedure, in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1, Springer Verlag, pp. 162–201. Originally appeared as [Prawitz 1960].
- ROBINSON G. AND WOS L. [1969], Paramodulation and theorem-proving in first order theories with equality, in B. Meltzer and D. Michie, eds, 'Machine Intelligence', Vol. 4, Edinburgh University Press, pp. 135–150.
- ROBINSON J. [1965], 'A machine-oriented logic based on the resolution principle', *Journal of the Association for Computing Machinery* **12**(1), 23–41.
- ROBINSON J. [1968], The generalized resolution principle, in B. Meltzer and D. Michie, eds, 'Machine Intelligence', Vol. 3, American Elsevier, N.Y., pp. 77–94.
- ROGAVA M. [1967], 'On sequential modifications of applied predicate calculi (in Russian)', *Zapiski Nauchnyh Seminarov LOMI* **4**, 175–189. English Translation in: *Seminars in Mathematics: Steklov Math. Inst. 4*, Consultants Bureau, NY-London, 1969, p.77–81.
- RUSINOWITCH M. [1991], 'Theorem proving with resolution and superposition: an extension of the Knuth and Bendix completion procedure as a complete set of inference rules', *Journal of Symbolic Computations* **11**, 21–49.
- SCHUMANN J. [1994], 'Tableau-based theorem provers: Systems and implementations', *Journal of Automated Reasoning* **13**(3), 409–421.
- SCHÜTTE K. [1960], *Beweistheorie* (in German), Springer Verlag.

- SHANKAR N. [1992], Proof search in the intuitionistic sequent calculus, in D. Kapur, ed., '11th International Conference on Automated Deduction', Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Saratoga Springs, NY, USA, pp. 522–536.
- SHOSTAK R. [1978], 'An algorithm for reasoning about equality', *Communications of the ACM* **21**, 583–585.
- SLAGLE J. [1974], 'Automated theorem-proving for theories with simplifiers, commutativity and associativity', *Journal of the Association for Computing Machinery* **21**(4), 622–642.
- SMULLYAN R. [1968], *First-Order Logic*, Springer Verlag.
- SNYDER W. AND LYNCH C. [1991], Goal-directed strategies for paramodulation, in R. Book, ed., 'Rewriting Techniques and Applications', Vol. 488 of *Lecture Notes in Computer Science*, pp. 150–161.
- STICKEL M. [1985], 'Automated deduction by theory resolution', *Journal of Automated Reasoning* **1**, 333–355.
- TAMMET T. [1996], A resolution theorem prover for intuitionistic logic, in M. McRobbie and J. Slaney, eds, 'Automated Deduction — CADE-13', Vol. 1104 of *Lecture Notes in Computer Science*, New Brunswick, NJ, USA, pp. 2–16.
- VEANES M. [1996], Uniform representation of recursively enumerable sets with simultaneous rigid *E*-unification, UPMail Technical Report 126, Uppsala University, Computing Science Department.
- VEANES M. [1997a], On Simultaneous Rigid *E*-Unification, PhD thesis, Uppsala University.
- VEANES M. [1997b], The undecidability of simultaneous rigid *E*-unification with two variables, in G. Gottlob, A. Leitsch and D. Mundici, eds, 'Computational Logic and Proof Theory. 5th Kurt Gödel Colloquium, KGC'97', Vol. 1289 of *Lecture Notes in Computer Science*, Vienna, Austria, pp. 305–318.
- VEANES M. [1998a], The relation between second-order unification and simultaneous rigid *E*-unification, Technical Report MPI-I-98-2-005, Max-Planck Institut für Informatik, Saarbrücken.
- VEANES M. [1998b], The relation between second-order unification and simultaneous rigid *E*-unification, in 'Proc. Thirteenth Annual IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, Indianapolis, Indiana, pp. 264–275.
- VODA P. AND KOMARA J. [1995], On Herbrand skeletons, Technical report, Institute of Informatics, Comenius University Bratislava.
- VORONKOV A. [1985], 'A proof search method (in Russian)', *Vychislitelnye Sistemy* **107**, 109–123.
- VORONKOV A. [1990a], LISS — the logic inference search system, in M. Stickel, ed., 'Proc. 10th Int. Conf. on Automated Deduction', Vol. 449 of *Lecture Notes in Computer Science*, Springer Verlag, Kaiserslautern, Germany, pp. 677–678.
- VORONKOV A. [1990b], A proof-search method for the first order logic, in P. Martin-Löf and G. Mintz, eds, 'COLOG'88', Vol. 417 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 327–340.
- VORONKOV A. [1992], Theorem proving in non-standard logics based on the inverse method, in D. Kapur, ed., '11th International Conference on Automated Deduction', Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Saratoga Springs, NY, USA, pp. 648–662.
- VORONKOV A. [1996a], On proof-search in intuitionistic logic with equality, or back to simultaneous rigid *E*-Unification, UPMail Technical Report 121, Uppsala University, Computing Science Department.
- VORONKOV A. [1996b], Proof search in intuitionistic logic based on constraint satisfaction, in P. Miglioli, U. Moscato, D. Mundici and M. Ornaghi, eds, 'Theorem Proving with Analytic Tableaux and Related Methods. 5th International Workshop, TABLEAUX '96', Vol. 1071 of *Lecture Notes in Artificial Intelligence*, Terasini, Palermo Italy, pp. 312–329.
- VORONKOV A. [1996c], Proof-search in intuitionistic logic based on the constraint satisfaction, UPMail Technical Report 120, Uppsala University, Computing Science Department. Updated March 11, 1996.

- VORONKOV A. [1996d], Proof search in intuitionistic logic with equality, or back to simultaneous rigid E -unification, in M. McRobbie and J. Slaney, eds, 'Automated Deduction — CADE-13', Vol. 1104 of *Lecture Notes in Computer Science*, New Brunswick, NJ, USA, pp. 32–46.
- VORONKOV A. [1998a], Herbrand's theorem, automated reasoning and semantic tableaux, in 'Proc. IEEE Conference on Logic in Computer Science (LICS)', IEEE Computer Society Press, pp. 252–263.
- VORONKOV A. [1998b], Herbrand's theorem, automated reasoning and semantic tableaux, UPMAIL Technical Report 151, Uppsala University, Computing Science Department.
- VORONKOV A. [1998c], 'Proof-search in intuitionistic logic with equality, or back to simultaneous rigid E -unification', *Journal of Automated Reasoning* 21, 205–231.
- VORONKOV A. [1998d], Simultaneous rigid E -unification and other decision problems related to Herbrand's theorem, UPMAIL Technical Report 152, Uppsala University, Computing Science Department.
- VORONKOV A. [1999], 'Simultaneous rigid E -unification and other decision problems related to Herbrand's theorem', *Theoretical Computer Science* 224, 319–352.
- VORONKOV A. [2000a], Deciding K using \mathcal{M} , in A. Cohn, F. Giunchiglia and B. Selman, eds, 'Principles of Knowledge Representation and Reasoning (KR'2000)', pp. 198–209.
- VORONKOV A. [2000b], How to optimize proof-search in modal logics: a new way of proving redundancy criteria for sequent calculi, in 'Proc. 15th Annual IEEE Symp. on Logic in Computer Science', Santa Barbara, California, pp. 401–412.
- VORONKOV A. [2001a], 'How to optimize proof-search in modal logics: new methods of proving redundancy criteria for sequent calculi', *ACM Transactions on Computational Logic* 1(4), 1–35.
- VORONKOV A. [2001b], 'Proof-search in intuitionistic logic based on the constraint satisfaction and related complexity problems', *Logic Journal of the IGPL* pp. 1–26. to appear.
- WANG H. [1960], 'Towards mechanical mathematics', *IBM J. of Research and Development* 4, 2–22.
- WEIDENBACH C. [2001], Combining superposition, sorts and splitting, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 27, pp. 1965–2013.
- WOS L., OVERBEEK R. AND LUSK E. [1991], Subsumption, a sometimes undervalued procedure, in J.-L. Lassez and G. Plotkin, eds, 'Computational Logic. Essays in Honor of Alan Robinson.', The MIT Press, Cambridge, MA, pp. 3–40.
- WOS L., ROBINSON G., CARSON D. AND SHALLA L. [1967], 'The concept of demodulation in theorem proving', *Journal of the Association for Computing Machinery* 14, 698–709.
- ZHANG H. AND KAPUR D. [1988], First-order theorem proving using conditional rewrite rules, in E. Lusk and R. Overbeek, eds, '9th International Conference on Automated Deduction', Vol. 310 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 1–20.

Calculi and inference rules

A

(*abc*) — atomic branch closure rule ... 638

B

(*bc ξ*) — branch closure rule
specialized to ξ 675

(*bls*) — basic right superposition 672

(*brs*) — basic left superposition 672

BS ξ — basic superposition calculus
specialized for ξ 672

BSE — basic superposition calculus
for rigid equations 655

C

(\wedge_ξ) — ξ -conjunction rule 677

E

(*eqs*) — equality solution rule 672

F

(*fac*) — factoring rule 677

I

I ξ — inverse method calculus 677

IK ξ — inverse method calculus 642

L

LJ $^=$ — sequent calculus for
intuitionistic logic 680

LJ $^=_c$ — intuitionistic calculus of
constraint sequents 686

LK $^=$ — sequent calculus 622

LK $^=$ — sequent calculus for
classical logic 623

LK $^=$ for formulas in negation
normal form 628

LK $^=$ using signed formulas 624

R

(*rbls*) — rigid basic left superposition
rule 655

(*rbrs*) — rigid basic right superposition
rule 655

(*reqs*) — rigid equality solution rule .. 655

(*rfac*) — rigid factoring rule 665

(*rpar*) — rigid paramodulation rule ... 666

(*rrefl*) — rigid reflexivity rule 665

(*rres*) — rigid resolution rule 665

RRP — rigid resolution calculus 665

S

S4 $^=$ — calculus for modal logic with equal-
ity 690

T

T ξ — tableau calculus 675

(*tbs*) — tableau basic superposition rule 663

TBSE — tableau basic superposition
calculus 664

(*te ξ*) — ξ -tableau expansion rule 674

(*teqs*) — tableau equality solution rule 664

TK — tableau calculus 638

TK $^=$ 661

(*tpar*) — tableau paramodulation rule 661

Index

Symbols

$E\theta$ — application of θ to E	614
$R \cdot C$ — constrained rigid equation	655
$\mathcal{T}_{\mathcal{F}}$ — set of ground terms in \mathcal{F}	615
$\mathcal{T}_{\mathcal{F}}(X)$ — set of terms in \mathcal{F} with variables in X	615
$[x_1 \mapsto t_1, \dots, x_n \mapsto t_n]$ — substitution notation	614
$\#$	637
\square — empty clause	614
$\square\Gamma$	690
$\Gamma[s]$ — set of formulas Γ with an occurrence of term s	616
Γ_{\approx} — set of equations in Γ	686
α -rule	623
\approx — equality predicate	614
β -rule	623
δ -rule	623
$\diamond\Gamma$	690
$\exists\varphi$ — existential closure of φ	614
\exists^* -fragment of intuitionistic logic	651
$\forall\varphi$ — universal closure of φ	614
γ -rule	623
\leadsto — one-step derivability in BSE ..	655
\leadsto^* — many-step derivability in BSE ..	655
\leftrightarrow_E — symmetric rewriting relation ..	640
\leftrightarrow_E^* — reflexive and transitive closure of \leftrightarrow_E	640
\mathcal{F} — function signature	615
$\mathcal{T}\mathcal{C}$ — notation for constrained tableaux ..	663
\equiv — equal by definition	614
$\sigma \leq \theta$ — σ is more general than θ ..	614
$\sigma \models C$ — σ satisfies C	686
$\sigma\theta$ — composition of σ and θ	614
σ_x — substitution obtained by redefinition of σ in x	642
\succ — reduction ordering	619
\succeq — reduction ordering	619
θ -subsumption	674
ε — empty substitution	614
$\varphi[s]$ — formula φ with an occurrence of term s	616
\vdash — derivability	624
ξ -branch	674
ξ -branch closure	675
ξ -clause	677
ξ -conjunction rule	677
ξ -name of a subformula	669
ξ -tableau	674
ξ -tableau expansion rule	674

$r[s]$ — term r with an occurrence of term s	616
$s = t$ — equality constraint	633
$s \neq t$ — disequation	615
$s \succ t$ — ordering constraint	633
$s \succeq t$ — ordering constraint	633

A

analytic proof-search	643
answer constraint	656
for a tableau branch	657
antecedent	622
in intuitionistic logic	679
application of substitution	614
atom	614
atomic branch closure rule	638

B

backward proof-search	643
basic blocking	678
basic folding	678
basic left superposition	672
basic simplification	678
basic subsumption	678
bottom-up proof-search	643
branch	637
Brand's transformation	628

C

CEE-transformation	633
clause	614
closed branch	638
closed formula	614
closed sequent	622
closure	671
composition of substitutions	614
conjunctive subformula	669
conjunctive superformula	669
constrained clause	634
constrained rigid equation	655
constrained tableau	663
constraint	633
in $\mathbf{LJ}_c^=$	686
constraint equality elimination ..	630, 633
constraint satisfiability in $\mathbf{LJ}_c^=$	686
constraint sequent	686
constraint simplification	656

D

derivability of a formula	624
---------------------------------	-----

derivation of a formula	624
derivation skeleton	684
direct proof-search	643
disequation	615
$\text{dom}(\theta)$ — domain of substitution θ ..	614
domain of substitution	614
dummy	645

E

<i>E</i> -unification	647
eigenvariable	623
eigenvariable condition	623
empty clause	614
empty substitution	614
empty tableau	637
equality axioms	615
equality constraint	633
equality solution rule	672
equation	615
equivalent constrained clause	634
equivalent constraints	633
in $\text{LJ}_c^=$	686
expansion rule	667
explicit definability property of $\text{LJ}^=$..	680
extended superposition	620

F

factoring rule	677
flat clause	630
flat form of a clause	630
formula	614
formula instantiation	651
forward proof-search	643
free variable	645
Free variable tableau reflexivity	661
free variable tableau system	661
free variables	
of a constraint	686
function reflexivity axiom	617
function substitution axioms	615

G

global methods	667
goal-oriented proof-search	643
ground instance of a constrained clause	634
ground term	614
grounding substitution	614

H

Herbrand skeleton problem	651
Herbrand theorem	
for free variable tableaux	639

I

increasing applications of	
paramodulation	619
indirect proof-search	643
initial closure	671
instance	614
invertible rule	683

L

lazy paramodulation	676
least conjunctive superformula	669
example	670
left-hand side of a rigid equation	646
lexicographic path ordering	622
lifting	617
link constraint	634
link variable	634
$\text{lit}(\Gamma)$	637
literal	614
local methods	667

M

maximal paramodulation	620
metavariable	645
mgu — most general unifier	615
of substitutions	642
mgu tableau replacement rule	660
minimal subset unifier	674
minus-normalization	645
modification method	630
monadic semi-unification	648
monadic simultaneous rigid	
<i>E</i> -unification	692
more general substitution	614
most general unifier	
of substitutions	642
multiset of literals on a branch	637

N

negation normal form	627
non-analytic proof-search	643
nonliftable derivation	617

O

ordered paramodulation	619
ordering constraint	633

P

paramodulation into variable	618
paramodulation rule	617
partial theory reasoning	660
precedence relation	622
predicate substitution axioms	616
prenex fragment of intuitionistic logic	651

R

reduction ordering	619
redundancy criteria	621
reflexivity axiom	615
regular derivation	
in $LJ^=$	680
in $LK^=$	624
relaxed paramodulation	676
right-hand side of a rigid equation ..	646
rigid E -unification	646
rigid basic left superposition	655
rigid basic superposition	
left	655
right	655
rigid equality solution rule	655
rigid equation	646
rigid equations on a tableau branch ..	653
rigid factoring	665
rigid paramodulation	648
rigid paramodulation rule	666
rigid reachability	691
rigid reflexivity rule	665
rigid resolution rule	665
rigid variable	647

S

satisfiable constraint	633
in $LJ^=$	686
satisfiable set of constrained clauses ..	634
saturation procedures	621
second-order unification	648
sequent	622
in intuitionistic logic	679
set of ξ -names	671
example	670
sign	623
signed formula	623
simplification	621
simplification ordering	619
simultaneous paramodulation	644
simultaneous rigid E -unification	647
skeleton instantiation problem ..651,	685
skeleton of a derivation	684
Skolem negation normal form	627
solution clause	672
solution to a constraint	633
solution to a rigid equation	646
solution to a system of rigid equations	647
solvable rigid equation	646
strict superposition	620
subformula property	641
subset unification	674
subset unifier	674

substitution	614
substitutively inconsistent	
branch	639
set of clauses	664
tableau	639
subsumption	621
succedent	622
in intuitionistic logic	679
superformula	669
superposition	620
symmetric version	631
symmetry axiom	615
symmetry elimination	631
system of rigid equations	646

T

tableau	637
tableau basic superposition	663
tableau equality solution rule	664
tableau expansion rules	638
tableau function reflexivity rule	661
tableau paramodulation rule	661
top unification	676
top-down proof-search	643
total theory reasoning	660
transitivity axiom	615
transitivity elimination	
in CEE-transformation	634
in modification method	631

U

universal closure	614
-------------------------	-----

V

V -paramodulation	648
V -resolution	648
$vars(E)$ — set of free variables of E ..	614
$vars(C)$ — free variables of C	686
variant	615
of a closure	671

W

weak most general unifier	642
wmg — weak most general unifier ..	642

Automated Reasoning in Geometry

Shang-Ching Chou

Xiao-Shan Gao

SECOND READERS: Dongming Wang and Hantao Zhang.

Contents

1	A history review of automated reasoning in geometry	709
2	Algebraic approaches to automated reasoning in geometry	712
2.1	Proving theorems in elementary geometries	712
2.2	Proving theorems involving inequalities	723
2.3	Proving theorems in differential geometry	728
2.4	Mechanical geometric formula derivation	730
3	Coordinate-free approaches to automated reasoning in geometry	732
3.1	Area method	732
3.2	Bracket algebra methods	733
3.3	Clifford algebra methods	734
3.4	Gröbner bases methods	734
4	AI approaches to automated reasoning in geometry	734
4.1	Gelernter's geometry machine	735
4.2	A geometry deductive database	736
4.3	Automated diagram generating	737
4.4	Issues in developing a prover based on AI approaches	738
5	Final remarks	740
5.1	Purposes of studying geometric reasoning	740
5.2	Further research directions	740
	Bibliography	741
	Index	749

HANDBOOK OF AUTOMATED REASONING

Edited by Alan Robinson and Andrei Voronkov

© 2001 Elsevier Science Publishers B.V. All rights reserved

In the past 20 years highly successful methods for geometry theorem proving and discovering have been developed. This chapter gives a brief account of these successful methods. We will use elementary and understandable examples to show the essence of the techniques, letting the reader consult the related references for more detailed issues underlying these techniques.

1. A history review of automated reasoning in geometry

Generally, there are two approaches to proving geometry theorems using computers: the artificial intelligence (AI) approach and the algebraic computation approach. The earliest work in geometry theorem proving by computer programs was done by Gelernter and his collaborators [Gelernter 1959]. It was based on the human simulation approach and has been considered a landmark in the AI area for its time. Vos and his collaborators used their powerful general-purpose resolution theorem prover to experiment with proving theorems in Tarski's axioms for elementary geometry [McCharen, Overbeek and Vos 1976]. In spite of the success and significant improvements [Gillmore 1970, Nevins 1976, Coelho and Perceira 1986, Koedinger and Anderson 1990, Quaife 1989, Balbiani and del Cerro 1995] with these methods, the results did not lead to the development of a *powerful* geometry theorem prover.

In the area of algebraic computation approach, the earliest work dates back to Hilbert. In his classic book [Hilbert 1971], Hilbert outlined a decision method for a class of constructive geometry statements in *affine geometry*. As Tarski pointed out, Hilbert's result "is closely connected with the decision method for elementary geometry, but has a rather restricted character".

In 1951, Tarski published a decision method for the theory of real closed fields, thus giving a decision method for what he called elementary geometry [Tarski 1951]. In spite of subsequent improvements by Seidenberg [Seidenberg 1954] and others, for years variations of Tarski's method remained impractical for proving non-trivial theorems in geometry. In 1974, Collins made an important contribution along the Tarski line [Collins 1975]. His cylindrical algebraic decomposition (CAD) algorithm is currently the best general algorithm of Tarski type. This method was implemented by Arnon, and several difficult algebra-geometry related problems were solved by Arnon's program [Arnon and Mignotte 1988, Arnon 1988]. Another major practical improvement of Collins' method has been made in [Collins and Hong 1991].

Practically, Davis appears to be the first to explore the algebraic approach to proving geometry theorems using the computer [Davis and Cerutti 1969]. His approach for the computer proof of Pappus' theorem is essentially the one described by Hilbert, but he did not provide a unifying mechanical way to do it.

A breakthrough in automated geometry theorem proving (AGTP) is made by Wu. Restricting himself to a class of geometry statements of *equality type*, Wu introduced a method in 1977 which can be used to prove quite difficult geometry theorems efficiently [Wu 1978]. Wu's work became known outside China mainly through the papers [Wu 1984c, Chou 1984], and the fact that over 130 theorems were proved by the method in [Chou 1984] was quite encouraging. Ko and Hussain [Ko and Hussain

1985], Wang and Hu [Wang and Hu 1987, Wang and Gao 1987], Gao [Gao 1990], Kapur and Wan [Kapur and Wan 1990] also succeeded in implementing theorem provers based on various modified version of Wu's method. Later it was clarified [Wu 1984a] that the algebraic tools needed in Wu's approach can be developed from Ritt's work in [Ritt 1950]. The algebraic aspect of this approach is now known as the Wu-Ritt's characteristic set (CS) method. It is now the case that hundreds of theorems in Euclidean and non-Euclidean geometries can be proved automatically by computer programs with Wu's method.

The success of Wu's method has revived interest in proving geometry theorems by computers. In particular, the application of the Gröbner basis (GB) method [Buchberger 1985] to the same class of geometry theorems that Wu's method addresses has been investigated. In 1985–1986, three groups ([Chou and Schelter 1986, Kapur 1986, Kutzler and Stifter 1986]) reported practical successes. A recent tutorial on the Gröbner basis method can be found in [Wang 1998b]. Other successful elimination methods for automated geometry theorem proving (AGTP) include the resultant approach [Yang, Zhang and Hou 1992], the gcd computation approach [Kalkbrenner 1995], the numerical example checking approach [Hong 1986, Zhang, Yang and Deng 1990, Wang 1988], the Brauer-Seidenberg-Wang approach [Wang 1995b] and the Dixon resultant approach [Kapur 1997].

Here we would like to remind the reader that Wu's method and the GB method can only deal with theorems involving equalities, but not inequalities. Theoretically, Collins' method can prove (or disprove) any elementary sentences in the Tarski geometry. Many researchers focused on developing more efficient algorithms for special classes of problems involving inequalities. Wu proposed a method to find the maximal or minimal values for a polynomial (pol) function under certain conditions using the CS method and the Lagrangian multiplier method [Wu 1992a]. The work in [McPhee, Chou and Gao 1994] is based on a combination of Wu's method and the CAD method. The work in [Dolzmann, Sturm and Weispfenning 1996] is based on quantifier elimination methods for equations with low degrees. Recently, Yang et al proposed the complete discriminant theory which is quite efficient in finding real roots classifications for univariate pol equations [Yang, Zhang and Hou 1996]. Yang also developed an inequality prover which has been used to prove more than one thousand interesting geometric inequalities including many new ones [Yang 1998].

At the same time, automated derivation of geometric locus equations and other geometric formulas was investigated [Wu 1986a, Wang and Gao 1987, Chou 1987, Chou and Gao 1990a, Wang 1991, Wang 1995c]. About 120 problems in geometry were solved in [Chou and Gao 1989]. Dixon resultant computation is used to derive geometric formulas in [Kapur, Saxena and Yang 1994]. Formula derivation is actually to find the manifold solutions of equation systems.

The above work is concerned with *elementary geometries* in Wu's sense, i.e., geometries in which no differentiation is involved. The CS method is also applicable to *differential pols* [Ritt 1938]. In [Wu 1979], Wu extended the CS method to prove theorems in *differential geometry*. Extensive computer experiments with this method for the theory of space curves were done in [Wu 1987c, Chou and Gao 1991]. The results are encouraging, and nearly one hundred non-trivial theorems in space

curve theory have been proved [Chou and Gao 1991]. In [Li 1995b], Wu's method was used to prove theorems of space surfaces. In [Ferro and Gallo 1990, Ferro and Gallo 1994], new methods for proving theorems in differential geometry based on the computation of the dimensions of zero sets were proposed. In [Wang 1995b], Brauer-Seidenberg's elimination theory is modified to prove theorems in space curve theory [Wang 1995b]. In [Li and Cheng 1998], a method based on vector calculation for AGTP in differential geometry is proposed, which is capable of producing proofs like those in the textbooks. There have also been several successful applications of the CS method to mechanics [Wu 1987b, Chou and Gao 1993b, Chou and Gao 1993c]; notably, automated proofs of Newton's laws from Kepler's laws were given. Computer experiments in automated formula derivation in differential geometry and mechanics were also discussed in these pieces of work.

All the above methods have the same character that they first transform geometric properties into equations in coordinates of the related points and then deal with these equations. The search for a vector based method for AGTP began in the mid-eighties, because it is believed that such a method would produce more elegant proofs. In the early-nineties, several successful vector approaches were proposed: the area method [Chou, Gao and Zhang 1993a], the vector method for constructive statements [Chou, Gao and Zhang 1993b], the Gröbner basis method [Stifter 1993], the bracket algebra method [Richter-Gebert 1995] and the Clifford algebra method [Li and Cheng 1996, Fèvre and Wang 1997, Wang 1998a]. Experiments show that proofs produced by these methods are generally shorter than those given by the coordinate methods. Of these methods, the area method uses pure geometric invariants, such as area, ratio of segments, Pythagorean difference, etc. The main advantage of this method is that each step of the generated proof has clear geometric meanings. The computer program based on the area method has produced proofs of more than 500 geometry theorems, some of which are even shorter than those given by geometry experts.

More recently, the AI approach has been revived to such an extent that it can solve hundreds of difficult geometry problems and produce multiple and shortest proofs for geometry theorems efficiently [Chou, Gao and Zhang 1996a, Chou, Gao and Zhang 1996c]. The AI approach is also used for automated generation of construction steps of geometric diagrams and successfully applied to many difficult geometric construction problems (e.g., the Apollonius Problem) [Gao and Chou 1998a].

Methods of automated reasoning in geometry have a wide range of applications, including kinetic analysis of robotics [Wu 1989b, Huang and Wu 1992, Kapur 1997, Yang, Fu and Zheng 1997], linkage design [Gao, Zhu and Huang 1998a], computer vision [Kapur and Mundy 1988, Gao and Cheng 1998, Wang 1998a, Bondyfalat, Mourrain and Papadopoulou 1999], intelligent CAD [Gao and Chou 1998a, Gao and Chou 1998b], intelligent CAI (computer aided instruction) [Gao, Zhu and Huang 1998b, Li and Zhang 1998], solid modeling [Wu 1993, Kapur 1997], etc. Several pieces of software originaed from this field have been published [Gao, Zhang and Chou 1998, Li and Zhang 1998, Richter-Gebert and Kortenkamp 1999].

Finally, we want to say that although we hope to cover all the work in the subject,

some existing work might be missed. Also, the reader may consult previous surveys on the similar subject [Wu 1992b, Buchberger, Collins and Kutzler 1995, Wang 1996b, Kapur 1997]. In particular, a report on AGTP provers can be found in [Hong, Wang and Winkler 1995].

The rest of this review is divided into four sections. Section 2 is a review of the algebraic approaches to AGTP. Section 3 is a review of coordinate free approaches to AGTP. Section 4 is a review of AI approaches to AGTP. Section 5 is a summary of this paper.

2. Algebraic approaches to automated reasoning in geometry

2.1. Proving theorems in elementary geometries

This is the most developed and successful area using the characteristic set (CS), the Gröbner basis (GB), and other elimination methods. Roughly speaking, the methods can address those *geometry statements of equality type*, for which, in their algebraic form, the hypotheses can be expressed by a set (conjunction) of equations

$$(2.1) \quad \begin{array}{l} h_1(y_1, \dots, y_m) = 0 \\ h_2(y_1, \dots, y_m) = 0 \\ \dots \\ h_r(y_1, \dots, y_m) = 0, \end{array}$$

and the conclusion is also a pol equation $c(y_1, \dots, y_m) = 0$, where the h 's and c are pols with coefficients in a *base field* K . Usually, we assume, $K = \mathbf{Q}$, the field of rational numbers. Thus the algebraic form of the geometry statement would be

$$\forall y[(h_1 = 0 \wedge \dots \wedge h_r = 0) \Rightarrow c = 0].$$

However, such formulas are usually not valid because most geometry statements are only valid under some non-degenerate (ndg) conditions. Let us look at two concrete examples to see the real situations.

2.1.1. Two examples and the simple version of Wu's method

2.1. EXAMPLE. Let $ABCD$ be a parallelogram, and E the intersection of the two diagonals AC and BD . Show that $AE = CE$ (Fig. 1).

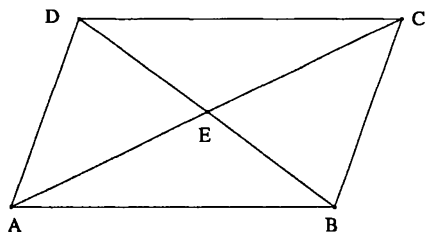


Fig. 1

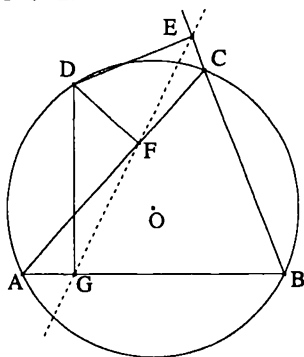


Fig. 2

The first step is to assign coordinates to the points, and then translate the hypotheses and conclusion of the statement into pol equations.

We can choose point A as the origin, and line AB as the x -axis of the coordinate system: $A = (0, 0)$, $B = (u_1, 0)$. We can assign the coordinates to point $C = (u_3, u_2)$. Since points A , B , and C can be arbitrarily chosen, their non-zero coordinates are independent variables or parameters, denoted by u 's. Once these three points are fixed, the other points D and E are determined; their coordinates are dependent variables, denoted by x 's. Let $E = (x_2, x_1)$, $D = (x_4, x_3)$.

Once we have the coordinate, the conversion of the hypotheses and conclusion to their algebraic forms is straightforward. Thus we have the following equations corresponding to the hypotheses:

$$\begin{aligned}
 (2.2) \quad & h_1 = u_1x_1 - u_1u_2 = 0 & DC \parallel AB \\
 & h_2 = u_2x_2 - (u_3 - u_1)x_1 = 0 & DA \parallel BC \\
 & h_3 = u_2x_4 - u_3x_3 = 0 & E \text{ is on } AC \\
 & h_4 = x_1x_4 - (x_2 - u_1)x_3 - u_1x_1 = 0 & E \text{ is on } BD.
 \end{aligned}$$

The conclusion $AE = CE$ can be expressed by $c = x_4^2 + x_3^2 - [(x_4 - u_3)^2 + (x_3 - u_2)^2] = 2u_3x_4 + 2u_2x_3 - u_3^2 - u_2^2 = 0$.

Thus the tentative algebraic form of the parallelogram theorem would be

$$\forall ux[(h_1 = 0 \wedge h_2 = 0 \wedge h_3 = 0 \wedge h_4 = 0) \Rightarrow c = 0].$$

The above formula is "almost" correct except for a ndg condition: it is not valid when A , B , and C are collinear.

A nice feature of Wu's method is that ndg conditions sufficient for a geometry statement to be valid can be generated automatically during the proof process. The basic operation of the method is pseudo division.

Pseudo Division. Let

$$\begin{aligned}
 g &= a_n y^n + \cdots + a_1 y + a_0 & (a_n \neq 0) \\
 f &= b_k y^k + \cdots + b_1 y + b_0 & (b_k \neq 0 \wedge k > 0)
 \end{aligned}$$

be two polys in the variable y , where the coefficients a_i , b_i are in $\mathbf{Q}[v_1, \dots, v_d]$, $y \notin \{v_1, \dots, v_d\}$. Here a_n (or b_k) and n (or k) are called the leading coefficient and leading degree of g (or f) in the variable y and denoted by $lc(g, y)$ and $ld(g, y)$ (or $lc(f, y)$ and $ld(f, y)$), respectively. We want to divide g by f in the variable y . If b_k

is 1, then we use long division from high schools; otherwise we use pseudo division. For our purpose, all we really care about is the pseudo remainder r , denoted by $r = \text{prem}(g, f, y)$, which verifies the following remainder formula:

$$(2.3) \quad b_k^s \cdot g = q \cdot f + r, \quad ld(r, y) < k,$$

where $s (\leq n - k + 1)$ is a non-negative integer.

The first step is to transform the set of hypotheses into a triangular set, where each equation introduces only one new dependent variable. Let us look at (2.2); h_1 introduces x_1 and h_2 introduces x_2 ; so far so good. But h_3 introduces two new dependent variables x_3 and x_4 at the same time; thus (2.2) is not in triangular form. However, it is easy to transform it into a triangular set by letting $f_1 = h_1$, $f_2 = h_2$, $f_3 = \text{prem}(h_4, h_3, x_4)$, $f_4 = h_4$. Then f_1, \dots, f_4 is a triangular set or an ascending chain ASC :¹

$$(2.4) \quad \begin{aligned} f_1 &= u_1 x_1 - u_1 u_2 = 0 \\ f_2 &= u_2 x_2 - (u_3 - u_1) x_1 = 0 \\ f_3 &= (-u_2 x_2 + u_3 x_1 + u_1 u_2) x_3 - u_1 u_2 x_1 = 0 \\ f_4 &= x_1 x_4 - (x_2 - u_1) x_3 - u_1 x_1 = 0. \end{aligned}$$

Now we can perform the key step of the method: the successive (pseudo) division of c , the conclusion pol, with respect to that triangular set, i.e.,

$$\begin{aligned} R_4 &= \text{prem}(c, f_4, x_4) = (2u_3 x_2 + 2u_2 x_1 - 2u_1 u_3) x_3 - (u_3^2 - 2u_1 u_3 + u_2^2) x_1 \\ R_3 &= \text{prem}(R_4, f_3, x_3) = (u_2 u_3^2 + u_2^3) x_1 x_2 - (u_3^3 - 2u_1 u_3^2 + u_2^2 u_3 - 2u_1 u_2^2) x_1^2 - \\ &\quad (u_1 u_2 u_3^2 + u_1 u_2^3) x_1 \\ R_2 &= \text{prem}(R_3, f_2, x_2) = (u_1 u_2 u_3^2 + u_1 u_2^3) x_1^2 - (u_1 u_2^2 u_3^2 + u_1 u_2^4) x_1 \\ R_1 &= \text{prem}(R_2, f_1, x_1) = 0. \end{aligned}$$

The last remainder R_1 is denoted by $\text{prem}(c; f_1, \dots, f_4)$ or $\text{prem}(c; ASC)$. In this particular case it turns out to be zero by computation. That means we have proved the theorem. To see this, first we always have the following *remainder formula* when doing a pseudo successive division of any pol g with respect to a triangular set $ASC = f_1, \dots, f_p$,

$$(2.5) \quad I_1^{s_1} \dots I_p^{s_p} g = Q_1 f_1 + \dots + Q_p f_p + R_1.$$

Here the I_k are leading coefficients of the f_k and $s_k \geq 0$ ($k = 1, \dots, p$); $R_1 = \text{prem}(c; ASC)$ is the final remainder. In this case, $p = 4$ and the I_k are

$$\begin{aligned} I_1 &= u_1 \\ I_2 &= u_2 \\ I_3 &= -u_2 x_2 + u_3 x_1 + u_1 u_2 \\ I_4 &= x_1. \end{aligned}$$

Since $R_1 = 0$ and $f_k = 0$ ($k = 1, \dots, 4$) by the hypotheses, the right side of (2.5) is zero. Thus the conclusion pol must be zero if we assume all $I_k \neq 0$ ($k = 1, \dots, 4$).

$I_k \neq 0$ are usually connected with non-degeneracy. Thus the last step is the analysis of subsidiary conditions $I_k \neq 0$. For example, $I_1 \neq 0$ and $I_2 \neq 0$ mean that

¹Strictly speaking, there are other restrictions for a triangular set to be an ascending chain [Wu 1984a].

points A , B , and C are not collinear; $I_3 \neq 0$ means that AC and BD have a normal intersection; $I_4 \neq 0$ means that D is not on line AB .

Before going to the next example, let us summarize this simple version of Wu's method [Wu 1978]:

Step 1. Assign coordinates to the points involved, then translate the geometric hypotheses to a set (conjunction) of pol equations $h_1 = 0, \dots, h_r = 0$; also the conclusion is a pol equation $c = 0$.

Step 2. Transform the set of hypothesis polys into a triangular set $ASC = f_1, \dots, f_p$. Generally, we have a complete triangular algorithm presented and referred to as Ritt's principle by Wu [Wu 1984a].

Step 3. Divide the conclusion pol c successively with respect to the triangular set $ASC = f_1, \dots, f_p$ to obtain the final remainder $R_1 = \text{prem}(c; ASC)$. If $R_1 = 0$, then the statement is confirmed under subsidiary conditions $I_k \neq 0$, where the I_k are the leading coefficients of the f_k .

Step 4. Analyze the subsidiary conditions $I_k \neq 0$ which are usually connected with non-degeneracy.

For this simple example, we don not have to use this general schema. We can solve the dependent variables x_1, \dots, x_4 successively in terms of the u 's, then substitute the solutions into the conclusion pol c to see whether it is identical to zero. This is exactly what *Hilbert's mechanical proof method* for constructive affine geometry statements does. However, Hilbert's original method does not provide ndg conditions which are important for a geometry statement to be valid. In the general case, we cannot solve dependent variables explicitly and have to use the above general schema. The following problem is such an example.

2.2. EXAMPLE (*Simson's Theorem*). D is a point on the circumscribed circle of triangle ABC . From D , perpendiculars are drawn to three sides BC , CA and AB . Let E , F and G be the three feet. Show that E , F and G are collinear (Fig. 2).

Step 1. Let $A = (0, 0)$, $B = (u_1, 0)$, $C = (u_3, u_2)$, $O = (x_1, x_2)$, $D = (x_3, u_4)$, $E = (x_5, x_4)$, $F = (x_7, x_6)$, and $G = (x_9, x_8)$. We then have a set of hypothesis equations H :

$$\begin{array}{ll}
 h_1 = 2u_1x_1 - u_1^2 = 0 & OA = OB \\
 h_2 = 2u_2x_2 + 2u_3x_1 - u_3^2 - u_2^2 = 0 & OA = OC \\
 h_3 = x_3^2 - 2x_1x_3 + u_4^2 - 2x_2u_4 = 0 & DO = OA \\
 h_4 = u_2x_5 - (u_3 - u_1)x_4 - u_1u_2 = 0 & E \text{ is on } BC \\
 (2.6) \quad h_5 = (u_3 - u_1)x_5 + u_2x_4 - (u_3 - u_1)x_3 - u_2u_4 = 0 & ED \perp BC \\
 h_6 = u_2x_7 - u_3x_6 = 0 & F \text{ is on } AC \\
 h_7 = u_3x_7 + u_2x_6 - u_3x_3 - u_2u_4 = 0 & FD \perp AC \\
 h_8 = u_1x_8 = 0 & G \text{ is on } AB \\
 h_9 = u_1x_9 - u_1x_3 = 0 & GD \perp AB.
 \end{array}$$

The conclusion that E , F , and G are collinear can be expressed by the equation $c = (x_6 - x_4)x_9 - (x_7 - x_5)x_8 + x_4x_7 - x_5x_6 = 0$.

Step 2. Let $f_4 = \text{prem}(h_5, h_4, x_5)$, $f_6 = \text{prem}(h_7, h_6, x_7)$, $f_i = h_i$ for $i \neq 4, 6$. We thus have a triangular set $ASC = f_1, \dots, f_9$:

$$\begin{aligned}
 f_1 &= 2u_1x_1 - u_1^2 \\
 f_2 &= 2u_2x_2 + 2u_3x_1 - u_3^2 - u_2^2 \\
 f_3 &= x_3^2 - 2x_1x_3 + u_4^2 - 2x_2u_4 \\
 f_4 &= (u_3^2 - 2u_1u_3 + u_2^2 + u_1^2)x_4 - (u_2u_3 - u_1u_2)x_3 - u_2^2u_4 + u_1u_2u_3 - u_1^2u_2 \\
 f_5 &= u_2x_5 - (u_3 - u_1)x_4 - u_1u_2 \\
 f_6 &= (u_3^2 + u_2^2)x_6 - u_2u_3x_3 - u_2^2u_4 \\
 f_7 &= u_2x_7 - u_3x_6 \\
 f_8 &= u_1x_8 \\
 f_9 &= u_1x_9 - u_1x_3.
 \end{aligned}
 \tag{2.7}$$

Step 3. Now use successive pseudo division to compute the final remainder $R_1 = \text{prem}(c; ASC) = 0$.

Since the final remainder R_1 is 0, by the remainder formula (2.5), $c = 0$ follows from $h_i = 0$ and subsidiary conditions $I_i \neq 0$, where the I_i are the leading coefficients of the f_i .

Step 4. Analysis of subsidiary conditions $I_k \neq 0$. Here the non-zerosness of I_1, I_2, I_8 , and I_9 mean that A, B , and C are not collinear. $I_7 \neq 0$ and $I_5 \neq 0$ are not necessary by a more careful analysis (Section 2.1.4). The role of the conditions $I_4 \neq 0$ and $I_6 \neq 0$ is very subtle, and they are *necessary* when using Wu's method or the GB method. See Example 2.3 below.

2.1.2. Geometry statements of constructive type

This simple use of Wu's method is powerful enough to prove hundreds of nontrivial geometry statements. But its application is restrictive. First, the ndg conditions $I_i \neq 0$ depend on the choices of the coordinates and this may cause problems. For instance, we may "prove" the 8_3 configuration problem using this method, which is actually invalid [Chou, Gao and Mcphee 1989]. Secondly, $I_i \neq 0$ are in algebraic form and there is no general method to transform these conditions into geometric form.

The great success of Wu's method is closely connected to some special geometry statements: the class of *constructive statements* [Wu 1984a, Wang and Hu 1987, Chou and Gao 1992]. Actually, the statements considered by Hilbert are constructive ones, but it has a rather restricted character: they are about those configurations formed by straight lines in a constructive way. Hilbert's method was clarified and revitalized by Wu [Wu 1982b], and extended later by Wang and Hu [Wang and Hu 1987] for a wider domain of application. In [Chou 1984, Chou and Gao 1992], a class of constructive geometry statements (henceforth referred to as Class C) is considered. Class C is actually the statements about the configurations which can be drawn by rulers and compasses. For instance, Examples 2.1 and 2.2 are in this class. About 85 percent of the 512 theorems in [Chou 1988] belong to this class.

The basic method introduced in Section 2.1.2 can be developed into a complete

method for constructive statements. For a constructive statement, point coordinates can be chosen automatically and coordinate-independent ndg conditions in geometric form can be generated. Furthermore, these ndg conditions are sufficient, i.e., a geometry statement is true in a complex geometry iff it is true under these ndg conditions [Chou and Gao 1992]. For geometry statements of constructive type, since new points are introduced one by one, the new dependent variables are introduced at most two by two. Therefore their corresponding algebraic problems are easier to solve than the algebraic problems corresponding to the general geometry problems.

More properties of constructive statements can be found in [Chou and Gao 1992]. In [Chou and Gao 1993e], most of the results about constructive statements have been extended to solid and Riemannian geometries.

2.1.3. Formulation problem

The above way of proving theorems is not the one adopted by most provers based on the logic approach. We start with a set of hypotheses which do not necessarily imply the conclusion and end up with the confirmation of the conclusion by adding some subsidiary conditions. Would it be possible that the original geometry statement is substantially weakened or changed? To answer this question we need to address the formulation problem – in what sense do we prove geometry theorems? Generally, for a geometry statement, the equality part of the hypotheses (i.e., $H = \{h_1 = 0, \dots, h_r = 0\}$) is easy to identify. However, the inequation part of the hypotheses, which is usually connected with non-degeneracy, is not so simple to identify. For a given geometry statement, a ndg condition that is obvious to one person might not be obvious to a second, and a third person might refuse to accept the condition as relevant or appropriate. The key issue here is how to understand and handle these ndg conditions.

Generally, for a geometric configuration defined by a set of equations

$$H = \{h_1(y_1, \dots, y_m) = 0, \dots, h_r(y_1, \dots, y_m) = 0\},$$

we want to decide whether an assertion $c(y_1, \dots, y_m) = 0$ on this configuration is valid. We define the notation

$$\text{Zero}(h_1, \dots, h_r) = \{(y_1, \dots, y_m) \in E^m \mid h_i(y_1, \dots, y_m) = 0 \text{ for } i = 1, \dots, r\},$$

where E is an extension of the base field \mathbf{Q} ; usually, $E = \mathbf{C}$ (the field of complex numbers; later we will discuss the case when $E = \mathbf{R}$, the field of really numbers). As we know the formula $\forall y \in E[H \Rightarrow c = 0]$, or equivalently

$$\text{Zero}(H) \subset \text{Zero}(c)$$

is usually not valid because of missing ndg conditions.

It is now accepted by most researchers that there are two different but related formulations for dealing with ndg conditions [Chou and Yang 1989, Kapur 1997].

Formulation F1. Identify some variables among y_1, \dots, y_m as parameters, then decide whether the conclusion $c = 0$ follows from the hypothesis H generically with

respect to those parameters. The zeros of the hypothesis pols can be decomposed into irreducible components as follows:

$$\text{Zero}(H) = V_1^* \cup \cdots \cup V_d^* \cup V_1^{\text{dege}} \cup \cdots \cup V_l^{\text{dege}},$$

where V_1^*, \dots, V_d^* are all those components on which the selected parameters are exactly those algebraically independent variables, corresponding to ndg cases; the others correspond to degenerate cases. If the conclusion is valid in all ndg cases, then we say the statement is *generically true*. If it is valid on none of the ndg cases, then it is *generically false*.

The concept of generically true was proposed by Wu [Wu 1984a]. The above description was given in [Chou 1988].

Formulation F2. Explicitly specify the ndg condition $D = \{d_1 \neq 0, \dots, d_q \neq 0\}$ as a part of the hypotheses. Then the aim is to decide whether the statement

$$(2.8) \quad \forall y[(H \wedge D) \Rightarrow c = 0]$$

is valid without adding any additional conditions.

Formulation F2 was proposed by Kapur using GB approach [Kapur 1988] and was adopted in [Chou and Gao 1990b, Kapur and Wan 1990, Ko 1988, Wang 1996a] using the CS approach.

In [Wang 1995c, Winkler 1990, Winkler 1992], other formulations are proposed.

F1 can help to find the missing ndg conditions. Furthermore, it addresses the nature of the statement: if a statement is proved to be generically false, it cannot be a theorem no matter how many reasonable additional ndg conditions are added. However, ndg conditions are usually in algebraic form and the currently used methods based on F1 usually generate ndg conditions more than needed. On the other hand, F2 is easier to understand. The geometry statement is exactly specified, and the user can select ndg conditions he/she thinks suitable. However, if the statement is proved to be false, we don't know the nature of the statement: whether it is generically false or the proof failed due to missing ndg conditions. The ndg conditions are often implicit in a statement in geometry textbooks and identifying them is sometimes very hard and subtle, even in Euclidean geometry.

Now we come to the question why E was chosen to be \mathbf{C} , the field of complex numbers, instead of \mathbf{R} , the field of real numbers. First, we emphasize that if the CS or GB method confirms a geometry statement according to either F1 or F2, then it is valid in *all* fields, including \mathbf{R} . However, if the statement is disproved by one of the two methods, then it is not valid in \mathbf{C} , but may be valid in \mathbf{R} . In axiomatic geometry, there are several systems of axioms. There is a system of axioms for *unordered metric geometry* which involves the relations incidence, perpendicularity, segment congruence, and angle congruence; \mathbf{C}^2 (or \mathbf{R}^2) is a *model* for the theory of this metric geometry [Wu 1984b]. If we want to decide whether a geometry statement (with universal quantifiers outside) is a *logical consequence* of the theory of metric geometry, then the CS and GB methods are complete, i.e., they are a *decision procedure* for such a theory.

2.3. EXAMPLE (*Simson's theorem continued*). Suppose that the only ndg condition is " A, B , and C are not collinear", i.e., $d_1 = u_1 u_2 \neq 0$ for Simson's theorem. Then we ask (according to F2) whether the statement is a logical consequence of the theory of metric geometry, or equivalently, in its algebraic form, whether

$$\forall ux \in C[(H \wedge d_1 \neq 0) \Rightarrow c = 0]$$

is valid. Then the CS or GB method proves it is not the case. That means the statement, as it is, is *not a theorem* (logical consequence) in the theory of metric geometry. Or putting it another way, the statement cannot be proved *without using axioms of order*. However, if we add another ndg condition that "the three sides of the triangle are non-isotropic (with non-zero the length)", the statement is true as verified by both methods. In Euclidean geometry, isotropic lines do not exist, thus Simson's theorem is proved by the CS or GB method under the sole ndg condition that A, B and C are not collinear.

2.1.4. The CS and GB methods

From the above discussion, we have four approaches: CS(F1), CS(F2), GB(F1), and GB(F2). According to Kapur, one can use direct or refutational approaches [Kapur 1997]. Thus altogether there are possibly $4 \times 2 = 8$ approaches. Here we cite the presentations for those approaches: for CS(F1), see [Wu 1984a, Chou 1988]; for CS(F2), see [Ko 1988, Chou and Gao 1990b, Kapur and Wan 1990, Wang 1995c]; for GB(F1), see [Chou and Schelter 1986, Kutzler and Stifter 1986, Chou 1988, Wang 1998b]; for GB(F2), see [Chou and Schelter 1986, Kapur 1988, Chou 1988, Kapur 1986, Wang 1998b]. We now briefly present a representative of each of the four approaches.

CS(F1). The method we just used for the two examples in Section 2.1.2 is actually based on Formulation F1. In both examples, the triangular sets ASC are irreducible and represent the only ndg component, V_1^* , and $prem(c; ASC) = 0$ means that $c = 0$ is valid on V_1^* , i.e., the statements are generically true.

CS(F2). Let S and G be two pol sets. Denote $Zero(S/G)$ the set difference $Zero(S) - \bigcup_{d \in G} Zero(d)$. Thus according to F2, the goal is to prove (2.8), i.e., to prove

$$Zero(H/D) \subset Zero(c),$$

where $H = \{h_1 = 0, \dots, h_r = 0\}$ and $D = \{d_1 \neq 0, \dots, d_q \neq 0\}$ are the equality part and the inequation part of the hypotheses, respectively.

Using Wu-Ritt's decomposition algorithm [Wu 1984a],

$$Zero(H/D) = \bigcup_{1 \leq i \leq k} Zero(PD(ASC_i)/D)$$

where the ASC_i are irreducible ascending chains;

$$PD(ASC) = \{g \mid prem(g, ASC) = 0\}.$$

To decide whether the statement is true or not (in C), we only need to verify whether $\text{prem}(c; \text{ASC}_i) = 0$ for all i . If we chose $D = \{u_1 u_2 \neq 0\}$ for the parallelogram example and D to be the same as in Example 2.3 for Simson's theorem, then $k = 1$, and ASC_1 are just the triangular sets ASC in (2.4) and (2.7). Since $\text{prem}(c; \text{ASC}_1) = 0$ in both examples (Step 3: successive division), they have been confirmed under the specified ndg conditions without adding any other conditions. **GB(F1)**. The conclusion $c = 0$ follows from the hypotheses $h_1 = 0, \dots, h_r = 0$ generically iff there is a non-zero pol U containing only the parameters such that $U \cdot c \in \text{Radical}(h_1, \dots, h_r)$. This is in turn equivalent to c being in the radical generated by h_1, \dots, h_r in the ring $\mathbb{Q}(u)[x]$, where the u are parameters, and the x are the dependent variables. This is equivalent to a Gröbner basis of $h_1, \dots, h_r, cz - 1$ in $\mathbb{Q}(u)[x]$ containing 1, where z is a new variable. This is the case for Simson's theorem as confirmed by a computer program based on this approach, and the pol U was also found during computing the GB. Thus under $U \neq 0$, $(H \Rightarrow c = 0)$. It is generically hard to interpret the geometric meaning of $U \neq 0$ automatically. However, for a constructive statement, if $(H \Rightarrow c = 0)$ is confirmed to be generically true, then $(H \Rightarrow c = 0)$ is valid under the *geometric* ndg conditions generated by the algorithm in [Chou and Gao 1992]. Based on this theorem, Simson's theorem has been proved by GB(F1) under the ndg conditions that the three vertices are not collinear and the three sides are non-isotropic.

GB(F2). First we observe that in any field, $d \neq 0$ iff $\exists z(zd - 1 = 0)$. Thus (2.8) is equivalent to

$$\forall y[\exists z_1 \cdots z_q (H \wedge d_1 z_1 - 1 = 0 \wedge \cdots \wedge d_q z_q - 1 = 0) \Rightarrow c = 0],$$

which is in turn equivalent (because c is free of z_i) to

$$\forall y z_1 \cdots z_q [(H \wedge d_1 z_1 - 1 = 0 \wedge \cdots \wedge d_q z_q - 1 = 0) \Rightarrow c = 0].$$

In algebraically closed fields, the above formula is equivalent to whether $H' = \{h_1, \dots, h_r, d_1 z_1 - 1, \dots, d_q z_q - 1, zc - 1\}$ generates the unit ideal, where the z are new variables. Thus the method is to compute a Gröbner basis of H' to see whether it contains 1. It is the case as confirmed by computers for the two examples. The use of new variables z_i and z was first introduced by Rabinowitsch in connection with a proof of Hilbert's Nullstellensatz. It has been used extensively in computer algebra, e.g., in [Gianni, Trager and Zacharias 1988]. In geometry theorem proving, it was first used in [Chou and Schelter 1986, Kapur 1986].

A large number of geometric problems are solved by programs based on these methods [Chou 1988, Wang and Gao 1987, Kutzler and Stifter 1986, Kapur 1986]. In [Chou 1988], extensive experiments were carried out for methods CS(F1), CS(F2) and GB(F1) using 512 geometric problems. It is the case that most of the theorems can be proved within seconds, and for most of the theorems ndg conditions in geometric form could be generated.

A typical example is Morley's trisector theorem [Wu 1984a, Chou 1988, Wang 1998b, Wang and Zhi 1998]: "The points of intersection of the adjacent trisectors of the angles of any triangle are the vertices of an equilateral triangle (Fig. 3)."

CS(F1) and GB(F1) confirmed it to be generically true, but with ndg conditions in algebraic form. This theorem has been proved under the ndg conditions that the three vertices are not collinear and the three sides are non-isotropic using CS(F2).

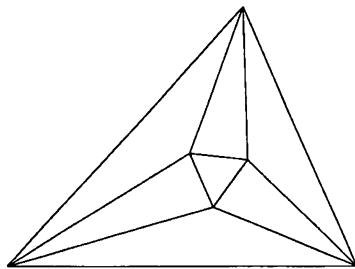


Fig. 3

The key factor to speed up the proving process is to have efficient implementations for the CS and GB methods. Many modifications of the original CS and GB methods are proposed for this purpose [Chou and Gao 1990b, Kapur and Wan 1990, Wang 1995a]. In particular, factorization of pols is proved to be quite important to enhance the speed for both CS [Chou and Gao 1990b] and GB methods [Wang 1998b]. For the CS method, factorization over extended field is a necessity. In [Wang 1996a, Wang and Zhi 1998], a new method of pol factorization is proposed and used to AGTP. The CS and GB methods are also used to solve the reducibility problems [Wu 1986b], to prove theorems in finite geometries [Lin and Liu 1992], to give transformation theorems of Cayley-Klein geometries [Chou and Ko 1986, Gao and Wang 1995], to prove theorems with complex numbers [Stokes 1990], to analysis robotics [Huang and Wu 1992, Wu 1989b, Kapur 1997, Yang et al. 1997], to design linkages [Wu 1989a, Gao, Zhu and Huang 1998a], to solve problems from computer vision [Kapur and Mundy 1988, Gao and Cheng 1998, Wang 1998a, Bondyfalat et al. 1999], to design intelligent CAD systems [Gao and Chou 1998a, Gao and Chou 1998b], to design intelligent CAI (computer aided instruction) systems [Gao, Zhu and Huang 1998b, Li and Zhang 1998], to solve problems from solid modeling [Wu 1993, Kapur 1997], etc. For details, please consult these references.

2.1.5. Other elimination methods and AGTP

Theoretically, any elimination method could be used to prove geometry theorems according to the two approaches. We singled out the CS and GB methods in the preceding section because they are the most extensively studied ones. In this section, we will give a brief introduction to other methods.

Wu's complete method needs pol factorization over algebraic extension fields which is a costly operation. In [Zhang et al. 1990], a complete method without using pol factorization was proposed. The method works as follows. Let g be a pol and f_1, \dots, f_r be a triangular set of pols. We define the resultant of g wrt f_1, \dots, f_r

inductively as

$$\text{resl}(g, f_1, \dots, f_r) = \text{resl}(\text{resultant}(g, f_r, \text{lv}(f_r)), f_1, \dots, f_{r-1})$$

where $\text{lv}(f_r)$ is the leading variable of f_r . It is known that if $\text{prem}(g, f_1, \dots, f_r) = 0$ then $g = 0$ follows from $f_1 = 0, \dots, f_r = 0$ generically and if $\text{resl}(g, f_1, \dots, f_r) \neq 0$ then $g = 0$ cannot be deduced from f_1, \dots, f_r . Otherwise, we have $\text{prem}(g, f_1, \dots, f_r) \neq 0$ and $\text{resl}(g, f_1, \dots, f_r) = 0$. In this case f_1, \dots, f_r must be reducible and the factors can be found in the computation procedure of the resultant. Repeating the above procedure, finally f_1, \dots, f_r can be factorized into ascending chains $\text{ASC}_1, \dots, \text{ASC}_t$ such that for each ASC_i , either $\text{prem}(g, \text{ASC}_i) = 0$ or $\text{resl}(g, \text{ASC}_i) \neq 0$, i.e., $g = 0$ is either generically true or generically false. Similar algorithms based on gcd computation were given in [Kalkbrener 1995].

The elimination in the CS method is bottom-up, i.e., eliminating the variables in an increasing order. A top-down elimination method was developed by Brauer in algebraic case [Brauer 1948] and extended to differential case by Seidenberg [Seidenberg 1955]. Recently, Wang showed that this technique can be used to give a zero decomposition of Wu-Ritt type and the efficiency of the method is quite good [Wang 1995b]. He also used this method to prove theorems in elementary and differential geometries. We now introduce the key idea of this elimination method. Let

$$(2.9) \quad P_1 = 0, \dots, P_r = 0, D \neq 0$$

be a pol equation system in variables x_1, \dots, x_n . Suppose that all of them involves x_n and P_1 has the lowest degree in x_n . Let $P_1 = Ix_n^d + U$ where U is of lower degree than d in x_n . Then (2.9) is equivalent to

$$I = 0, U = 0, P_2 = 0, \dots, P_r = 0, D \neq 0$$

and

$$P_1 = 0, R_2 = 0 \dots, R_r = 0, ID \neq 0$$

where $R_i = \text{prem}(P_i, P_1)$. Continue the above process, we can eliminate all variables and obtain a series of triangular sets. Both the direct approach and the refutational approach can be used to prove theorems with this method.

Both the CS and the GB methods eliminate variables one by one. In [Kapur et al. 1994], the concept of the Dixon resultant was extended to give an efficient method to eliminate multiple variables simultaneously. The Dixon resultant was used to prove geometry theorems using both direct and refutational approaches according to two formulations [Kapur and Saxena 1995, Kapur 1997]. We now give a brief introduction to the Dixon resultant method. Let P_1, \dots, P_{n+1} be pols in n variables x_1, \dots, x_n with coefficients as pols in parameters u_1, \dots, u_k . Let $\bar{x}_1, \dots, \bar{x}_n$ be n new variables. The *Cancellation matrix* of P_i is defined to be the following matrix:

$$C_P = \begin{bmatrix} P_1(x_1, \dots, x_n) & \cdots & P_{n+1}(x_1, \dots, x_n) \\ P_1(\bar{x}_1, \dots, x_n) & \cdots & P_{n+1}(\bar{x}_1, \dots, x_n) \\ \vdots & \cdots & \vdots \\ P_1(\bar{x}_1, \dots, \bar{x}_n) & \cdots & P_{n+1}(\bar{x}_1, \dots, \bar{x}_n) \end{bmatrix}$$

Since for all $1 \leq i \leq n$, $x_i - \bar{x}_i$ is a zero of C_P , $\prod_{i=1}^n (x_i - \bar{x}_i)$ divides $|C_P|$. The *Dixon pol* of P_i is defined as

$$\delta_P = \frac{|C_P|}{\prod_{i=1}^n (x_i - \bar{x}_i)}.$$

The main property of the Dixon resultant is that $\delta_P = 0$ is a necessary condition for $P_1 = 0, \dots, P_{n+1} = 0$ to have zeros. It often happens that $|C_P| = 0$ and we cannot get any information. In [Kapur et al. 1994], a method of rank submatrix construction is proposed to solve this problem. The details of the method and its application to AGTP can be found in [Kapur et al. 1994, Kapur 1997].

2.1.6. Proving geometry theorems by numerical computation

Based on the CS method, Hong showed that to prove a geometry statement, only a single numerical example is needed to check [Hong 1986]. To understand the method, let us mention the simple fact: a pol $p(x_1) \in Q[x_1]$ is identically zero if $p(x_0) = 0$ for a sufficiently large rational number x_0 . Hong's work is actually a generalization of the above result. Hong's work is generalized and used to test whether an algebraic variety is included in another variety in [Wang 1988].

In [Zhang et al. 1990], a method of proving geometry theorems by checking several numerical examples instead of one is presented. This method is similar to Hong's method and is based on a generalization of the following fact: a pol $p(x_1)$ of degree d is identically zero if it has more than d distinct roots.

In both of the above methods, approximate calculations are needed and at last we need to check whether an approximate number is small enough to be zero which is a difficult problem. However, in the case of linear geometry statements, the approximation problem can be avoided by using rational number calculation which is widely available in the symbolic computer software. A prover for linear statements has been developed and has been used to prove many nontrivial examples [Yang et al. 1992].

In [Ferro, Gallo and Gennaro 1998, Rege 1995], probabilistic methods for AGTP were studied.

2.2. Proving theorems involving inequalities

The methods reviewed in Section 2.1 address geometry statements of equality type and are complete only for geometry over complex numbers. If a geometry statement is proved in complex geometry, it is also valid in Euclidean geometry. The converse

is not true. An example which is valid in \mathbf{R} , but not in \mathbf{C} , the field of complex numbers, is the 8_3 problem [Kutzler 1989, Chou et al. 1989, Conti and Traverso 1995, Wang 1995c]. A simple algebraic example is $\forall u_1 x_1 [u_1^2 + x_1^2 = 0 \Rightarrow x_1 = 0]$. Both the CS and GB methods can disprove this formula in \mathbf{C} , but cannot confirm it in \mathbf{R} . However, such kinds of formulas rarely appear in Euclidean geometry. If we change the above formula a “little” bit: $\forall u_1 x_1 [u_1^2 + x_1^2 - 1 = 0 \Rightarrow x_1 = 0]$, then for $u_1 \in (-1, 1)$, x_1 always has solutions in \mathbf{R} . Such kinds of formulas are called \mathbf{R} -generic. If an \mathbf{R} -generic formula is not valid in \mathbf{C} , it is also not valid in \mathbf{R} . Most geometry statements of equality type are \mathbf{R} -generic, and for such statements the CS and GB methods are complete for Euclidean geometry [Chou and Yang 1989]. This is *the real reason* that these methods, which are complete only for complex geometry, can prove so many theorems in real geometry. But for theorems involving inequalities, we still need to develop new methods.

2.2.1. Proving theorems by quantifier elimination

Theoretically, Collins’ method can prove (or disprove) any first order statements in the Tarski geometry. In [Arnon 1988], the CAD method is used to geometry theorems in an interactive way. Thanks to the generosity of Collins and Hong, we have been able to experiment with proving geometry theorems using Hong’s implementation of Collins’ CAD algorithm. The results were very encouraging. We use the following example to give some idea about how far Collins’ method can reach now.

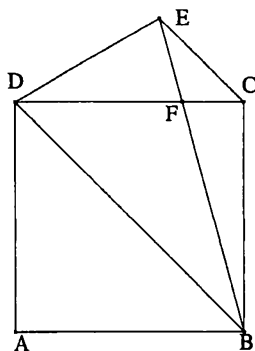


Fig. 4

2.4. EXAMPLE. Let $ABCD$ be a square. CE is parallel to the BD such that $BE = BD$. F is the intersection of BE and DC . Show that $DF = DE$ (Fig. 4).

Let $A = (0, 0)$, $B = (u_1, 0)$, $C = (u_1, u_1)$, $D = (0, u_1)$, $E = (x_1, x_2)$ and $F = (x_3, u_1)$. Then the hypotheses can be expressed by the following equations:

$$h_1 = x_2^2 + x_1^2 - 2u_1x_1 - u_1^2 = 0$$

$$h_2 = u_1x_2 + u_1x_1 - 2u_1^2 = 0$$

$$h_3 = x_2x_3 - u_1x_2 - u_1x_1 + u_1^2 = 0$$

$$BE = BD$$

$$CE \parallel BD$$

$$F \text{ is on } BE.$$

The conclusion ($DF = DE$) can be expressed by $c = (x_3 - 0)^2 + (u_1 - u_1)^2 - [(x_1 - 0)^2 + (x_2 - u_1)^2] = x_3^2 - x_2^2 + 2u_1x_2 - x_1^2 - u_1^2 = 0$.

Thus the algebraic form of the above statement is:

$$\forall u_1 x_1 x_2 x_3 [(h_1 = 0 \wedge h_2 = 0 \wedge h_3 = 0 \wedge u_1 \neq 0) \Rightarrow c = 0],$$

which was proved to be valid by Hong's program in 16 seconds on a SPARC-20 Station. This theorem has been considered fairly difficult in high school geometry. Also the previous implementation of Collins' method (Arnon's program) was unable to prove this theorem within reasonable computer resources. Collins' method was also used to prove this theorem in [Wang 1991].

In the area of AGTP, Collins' method requires further improvements in order to prove a substantial number of non-trivial theorems in practice.

In [Dolzmann et al. 1996], a quantifier elimination algorithm for linear and quadratic equations is presented. A *generic quantifier elimination method* is also proposed. In this method, variables are divided into parameters and variables, and pure parametric expressions are assumed to be non-zero. These expressions are similar to the ndg conditions in Wu's method. The program based on this method has been used to prove a large number of difficult geometry theorems. One reason behind the success of this method is that algebraic equations for most geometry theorems only involve quadratic equations. In [Weispfenning 1994], this method is used to solve many computational geometry problems. In [Dolzmann 1998], quantifier elimination methods are used to solve the real implicitization of the Enneper surface.

2.2.2. Proving theorems by optimization

Based on his CS method, Wu proposed a method to solve the following problem [Wu 1992a].

Problem Ineq. Let $\mathbf{R}^n(\mathbf{X})$ be the *real* Euclidean space of dimension n in the coordinates $\mathbf{X} = (x_1, \dots, x_n)$ and D a domain in \mathbf{R}^n . Let f, h_i ($i \in I = \{1, \dots, r\}, r < n$) and g be all real pols in $\mathbf{R}^n[\mathbf{X}]$ over the domain D . To determine for what real value c we shall always have

$$f \geq c \text{ or } f > c \text{ or } f \leq c \text{ or } f < c$$

under the conditions

$$\mathbf{HS} = 0, \text{ where } \mathbf{HS} = \{h_i \mid i \in I\} \text{ and } g \neq 0.$$

Wu proved the following *finite kernel theorem*. Let \mathbf{HS} be an arbitrary pol-set and f an arbitrary pol in $\mathbf{R}[\mathbf{X}]$. Then we can construct a *finite* set of real values K such that the extremal values of f under the constraint $\mathbf{HS} = 0$ is contained in K .

To solve this problem, Wu uses the *Lagrangian pol* with *Lagrangian multipliers* $\lambda_j, j \in M$ $L = f + \sum_{j \in M} \lambda_j h_j$. The *Lagrangian pol-set* is

$$\mathbf{LS} = \left\{ \frac{\partial L}{\partial x_i}, h_j \mid i \in N, j \in M \right\}.$$

The *Jacobian* for $t = (i_1, \dots, i_m) \in T$ is the determinant $J_t = \frac{\partial(h_1, \dots, h_m)}{\partial(x_{i_1}, \dots, x_{i_m})} = \left| \frac{\partial h_j}{\partial x_{i_k}} \right|$. The *Jacobian pol-set* is the pol-set

$$\mathbf{JS} = \{J_t, h_j \mid t \in T, j \in M\}.$$

Then the points where extremal values are achieved are contained in the projection of the zeros of the Lagrangian pol-set and the Jacobian pol-set. Based on this observation, using the CS method, we can compute the finite kernel.

2.5. EXAMPLE. [Wu 1995] Suppose that two cars of given form are moving respectively along the X -axis and the Y -axis in positive directions with known velocities $v_1 > 0$ and $v_2 > 0$. To decide whether the cars will collide or not, and to determine in the colliding case the time and place of first collision.

Let us consider the case in which the two cars are both of elliptical form given by $(c_1 < 0, c_2 < 0)$ $f_1(x, y) = b_1^2(x - c_1)^2 + a_1^2y^2 - a_1^2b_1^2$, $f_2(x, y) = b_2^2x^2 + a_2^2(y - c_2)^2 - a_2^2b_2^2$. Then the collision problem is seen to be a Problem Ineq for which $D = \{t > 0\} \subset \mathbb{R}^3(t, x, y)$, $f = t$, $g = 1$, $h_1 = f_1(x - v_1t, y)$, $h_2 = f_2(x, y - v_2t)$. In the case of having *generic* values for a_i, b_i, c_i and v_i , Wu's method gives rise to an irreducible pol equation of degree 8 in t with 696 terms. Leaving aside some uninteresting cases, the two cars will collide if and only if this equation has a positive root. The time and place of first collision can be easily determined if numerical values of a_i, b_i, c_i and v_i are substituted.

The method is used to prove geometry theorems involving inequalities [Wu 1992a], to prove trigonometric inequalities [Wang 1993], to solve non-linear programming problems [Wu 1995], to solve optimization problems [Wu 1992a], etc.

2.2.3. AGTP by combining the CS method and the CAD method

A theorem involving inequalities generally also involves equalities. Since the CS and GB methods work so well for equality problems, we might expect a combination of the CS (or GB) method with Collins' method could solve problems not in the scope of the CS method, but which cannot be solved by Collins' method alone within the available time and space. The work in [Chou, Gao and Arnon 1992] is in this direction, and a number of hard problems were solved with some human interaction. Here we use the following simple example to illustrate the basic idea.

2.6. EXAMPLE. Let $ABCD$ be a parallelogram. Show that points B and D are on either side of diagonal AC .

This "trivial" fact is repeatedly used in traditional proofs of the parallelogram theorem. However, it seems nontrivial to find a rigorous traditional proof of this fact. (Try it!)

Let $A = (0, 0)$, $B = (u_1, 0)$, $C = (u_2, u_3)$, and $D = (x_2, x_1)$. Then we have two equations for the hypotheses

$$h_1 = u_1x_1 - u_1u_3 = 0$$

$$AB \text{ is parallel to } CD$$

$$h_2 = u_3x_2 - (u_2 - u_1)x_1 = 0$$

AD is parallel to BC .

The conclusion that B and D are on either side of AC is $g < 0$, where $g = (u_3u_1 - u_2 \cdot 0)(u_3x_2 - u_2x_1) = u_1u_3^2x_2 - u_1u_2u_3x_1$. We want to decide whether the following statement is valid under certain ndg conditions: $\forall u_1u_2u_3x_1x_2[(h_1 = 0 \wedge h_2 = 0) \Rightarrow g < 0]$. Here u_1, u_2, u_3 are selected to be parameters, and x_1 and x_2 are selected to be dependent variables. Reducing g to canonical form modulo the ideal (in $\mathbf{Q}(u)[x]$) generated by h_1 and h_2 , we obtain $g = -u_1^2u_3^2$. This canonical form of g modulo the ideal is only valid under the conditions $u_1 \neq 0$ and $u_3 \neq 0$; in other words, u_1 and u_3 occur in the denominators of the elements c_1 and c_2 of $Q(u)[x]$ such that $g = c_1h_1 + c_2h_2$. Thus we have $g < 0$, under the condition that $u_1u_3 \neq 0$. Note that $u_1u_3 \neq 0$ is indeed connected with non-degeneracy, i.e. to insure that points A, B and C are not collinear.

A more general scheme has been proposed, and it has been used to solve the 8_3 problem automatically [McPhee et al. 1994]. Recently, N. MCPhee gives an automatic solution to the Steiner-Lehmus theorem and the Pompieu's theorem based on a combination of the CS method and Collins' CAD method [McPhee et al. 1994].

2.2.4. Complete discriminant systems and AGTP

In [Yang, Hou and Zeng 1996] a powerful tool, called the *complete discrimination system* (CDS) was introduced, which can be considered as an extension of the Sturm theorem. For a univariate pol equation $P(x) = 0$ of degree n , the CDS can be used to give the conditions that $P(x) = 0$ has p and q distinct real and complex solutions respectively and the multiplicities for these solutions are r_1, \dots, r_p and c_1, \dots, c_p such that $\sum_{i=1}^p r_i + \sum_{j=1}^q c_j = n$.

By means of CDS, together with Wu's method and a partial CAD algorithm, a generic program called "EXPLORER" was implemented in Maple that is able to discover and prove new inequalities [Yang, Hou and Xia 1998]. Using this program, Yang et al have re-discovered 37 inequalities in the first chapter of the monograph on geometric inequalities [Mitrinovic, Pecaric and Volenec 1989]. One of the inequalities "discovered" in this way is about the "basic inequality of triangles." For a triangle, from the basic inequalities about the three sides $a + b < c$, $a + c < b$, and $b + c < a$, the program can discover the basic inequality about the half perimeter s , circumradius R , and inradius r of the triangle:

$$s^4 + 2r^2s^2 - 4R^2s^2 - 20rRs^2 + 12r^3R + 48r^2R^2 + r^4 + 64rR^3 \leq 0.$$

An interesting inequality about triangles is discovered in [Guergueb, Mainguené and Roy 1998]. It would interest to know if this inequality can be discovered automatically.

In most of the geometric inequalities about triangles, there are radicals. A dimension-decreasing algorithm introduced by Yang [Yang 1998] can treat these kinds of inequalities efficiently. Based on this algorithm, a generic program called *BOTTEMA* was implemented on a PC computer. More than 1000 algebraic and geometric inequalities including hundreds of open problems have been verified in this way. The total CPU time spent for proving 120 basic inequalities from Bottema's

monograph, “Geometric Inequalities” on a Pentium/200, was 20-odd seconds only.

2.3. Proving theorems in differential geometry

An advantage of the CS method is that it can be extended to cover ordinary and partial algebraic differential equations [Ritt 1950]. As a consequence, theorems from differential geometry and mechanics can be proved automatically [Wu 1987a].

2.3.1. Space curves and mechanics

Here we are dealing with differential pol rings over a differential field (usually it is $\mathbb{Q}(t)$) in which there is a third operation, “ $'$ ”, compatible with the two operations of a ring, “ $+$ ” and “ \cdot ”:

$$(a + b)' = a' + b', (ab)' = a'b + ab'.$$

The pseudo division, triangular algorithm, and the variety decomposition algorithm can be extended to the differential pol case with minor modifications [Ritt 1950, Wu 1987a, Chou and Gao 1993a].

The geometry statements addressed are still of *equality type*. In the local theory of space curves, one uses parametric representation of a curve: $C = (x(t), y(t), z(t))$. The practical problems encountered in the curve theory and mechanics are of the similar nature. Thus we use the following elegant example first worked by Wu [Wu 1987b] to illustrate the type of problems we address:

2.7. EXAMPLE (*The Kepler–Newton problem*). Kepler’s first two laws are:
(K1) The planets move in elliptic orbits with the sun as a focus (Fig. 5).
(K2) The vector from the sun to the planet sweeps equal areas in equal times.
Newton’s law of gravitation (special form):
(N1) The acceleration of a planet is inversely proportional to the square of the distance from the sun to the planet.
Now we want to prove that (K1) and (K2) imply (N1).

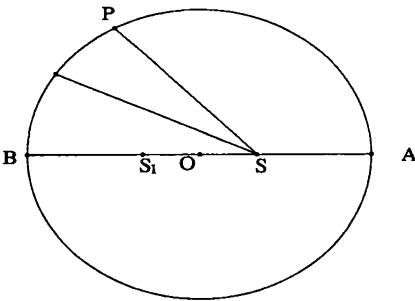


Fig. 5

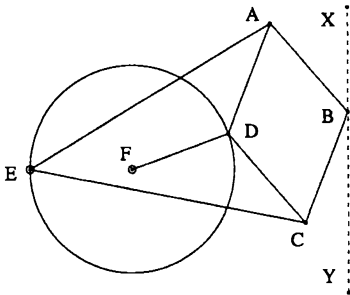


Fig. 6

Choose the sun as the origin of the coordinate system, and let $(x(t), y(t))$ be the position of the planet; $(-c, 0)$ be the center of the ellipse; h be the area velocity of the planet. Then the equation part of the hypotheses is:

$$\begin{array}{ll}
K_{11} : & \frac{(x+c)^2}{a^2} + \frac{y^2}{b^2} - 1 = 0 \\
K_{12} : & a^2 - (b^2 + c^2) = 0 \\
K_2 : & x'y - xy' - h = 0, \quad \text{with } h' = 0. \\
f_1 : & r^2 - (x^2 + y^2) = 0 \\
f_2 : & A^2 - (x''^2 + y''^2) = 0.
\end{array}$$

The conclusion N_1 is: $[Ar^2]' = 0$.

Then a tentative algebraic form for this problem would be:

$$\forall v[(K_{11} \wedge K_{12} \wedge K_2 \wedge f_1 \wedge f_2) \Rightarrow N_1].$$

As usual, x' denotes the derivative of x with respect to t . Here, a , b , c , and h are constants (independent of t). Again, the above formula is valid under some ndg conditions. With the simple use of the CS method described in Section 2.1.2, Wu proved a variation of the above specification [Wu 1987b] under some ndg conditions. As in elementary geometry, there are also two Formulations F1 and F2 for proving theorems in differential geometry. While these two formulations and the CS methods for them are similar to the case of elementary geometry, Formulation F1 is much more complicated than a simple rewording [Chou and Gao 1993c]. Nearly 100 theorems in space curves and 10 statements in mechanics have been proved [Chou and Gao 1991, Chou and Gao 1993d].

In [Ferro and Gallo 1990, Ferro and Gallo 1994], methods for proving theorems in differential geometry based on the computation of the dimensions of zero sets were proposed. It tries to find components with the highest dimension and prove the conclusion on these components. One problem with this approach is that the component with the highest dimension might not be the main component.

Another elimination method that works in differential case is the Brauer-Seidenberg technique. Since this technique can be used to eliminate quantifiers, it actually provides a decision method for differential closed field. Wang has modified this method to give a zero decomposition theorem and used it to prove theorems in differential geometry [Wang 1995b].

2.3.2. Space surfaces

This involves partial differential pols (pdp). In this area, Wu has developed tools [Wu 1979, Wu 1982a, Wu 1987a, Wu 1989c] based on the work [Ritt 1950, Thomas 1954, Cartan 1946], where again the method is only for geometry statements of equality type. This is similar to, but much more complicated than the ordinary differential pol case. Some experiment results are given and a new result is discovered in [Li 1995b]. The method currently used is the simple one described in Section 2.1.2, i.e., first triangularize a pdp set to obtain a pdp ascending chain ASC , then do similar successive pseudo divisions of the conclusion pdp with respect to that ASC to see whether the final remainder is zero. In elementary geometry such simple use of the CS method often proves a geometry statement to be generically true because most statements are of constructive types. However, in the pdp case,

the situation is unclear. There is no work to formulate "generically true" precisely as in Formulation F1 for elementary geometry. Formulation F2 is straight forward and the CS method for F2 can presumably be given, though no experimental work has been done yet.

2.3.3. Clifford algebra approach for AGTP

In differential geometry textbooks, vector algebra and Frenet moving frames are used to solve problems in the local theory of space curves. Li and Cheng recognized that Clifford algebra is more suitable for symbolic vector equations solving than vector algebra. Based on Wu's method of characteristic sets, they proposed a method that employs Clifford algebraic representation for geometric entities and constraints [Li and Cheng 1998]. A prover based on this method is capable of producing proofs much the same with those used in the textbooks.

The procedure of proving a theorem is composed of three stages: (a) find a reduction set, (b) find a parametric reduction set, and (c) find a characteristic set. To compute a reduction set, for scalar equations, Wu's method is used; for vector equations, the Clifford algebraic reduction method is used. For equations of differential forms, these elimination techniques can be used to compute a triangular set and to prove theorems about surfaces similar as in Section 2.1.1. This vector approach is not a decision procedure. In the final stage, coordinates are needed in order to give a complete method.

To overcome the difficulty of integrability pols in computing a characteristic set in the local theory of space surfaces, Li also proposed a simple method [Li 1995a] to integrate the calculus of differential forms with Wu's method.

2.4. Mechanical geometric formula derivation

2.4.1. Elementary geometry

There are two kinds of problems in elementary geometry other than theorem proving. One is finding locus equations, the other is deriving geometry formulas. Automatic derivation of geometry formulas were studied in [Wu 1986a, Chou 1984, Wang and Gao 1987, Chou 1987, Chou and Gao 1990a, Wang 1995c, Kapur et al. 1994]. We use Heron's Formula to illustrate this type of problems.

2.8. EXAMPLE. Find the formula for the area of a triangle ABC in terms of its three sides.

Let a, b , and c be the three sides, $B = (0, 0)$, $C = (a, 0)$, and $A = (x_1, x_2)$. Then the geometry conditions can be expressed by the following set of pol equations:

$$\begin{aligned} h_1 &= x_2^2 + x_1^2 - 2ax_1 - b^2 + a^2 = 0 & b &= AC \\ h_2 &= x_2^2 + x_1^2 - c^2 = 0 & c &= AB \\ h_3 &= ax_2 - 2k = 0 & k &= \text{the area of } ABC. \end{aligned}$$

The aim is to find a pol equation involving only a, b, c , and k which is a consequence of the above equations and some ndg conditions.

In general, for a geometric configuration given by a set of pol equations $h_1(u_1, \dots, u_q, x_1, \dots, x_p) = 0, \dots, h_r(u_1, \dots, u_q, x_1, \dots, x_p) = 0$ (possibly with a set of inequations $\{D = d_1(u, x) \neq 0, \dots, d_s(u, x) \neq 0\}$), we want to find a relation (formula) between arbitrarily chosen variables u_1, \dots, u_q (parameters) and a dependent variable, say, x_1 . In [Chou and Gao 1990a], CS and GB methods are used for formula derivation. In [Wang 1991], CS, GB and Wang's methods are used for formula derivation. In [Kapur et al. 1994], Dixon resultant is used for formula derivation. Heron's formula can be easily derived by any of the above methods:

$$16k^2 + c^4 - (2b^2 + 2a^2)c^2 + b^4 - 2a^2b^2 + a^4 = 0.$$

Here is a more interesting example.

2.9. EXAMPLE (Peaucellier's Linkage). Links AD , AB , DC and BC have equal length, as do links EA and EC . We assume $FD = EF$. The locations of joints E and F are fixed points on the plane, but the linkage is allowed to rotate about these points. As it does, what is the locus of the joint B ? (Fig. 6)

Let $F = (0, 0)$, $E = (r, 0)$, $C = (x_2, y_2)$, $D = (x_1, y_1)$, $B = (x, y)$, n and m be the lengths of the projections of CD and BC on BD and AC when E, D, B are collinear. Then the geometry conditions can be expressed by the following set of equations H

$$\begin{aligned} h_1 &= y_1^2 + x_1^2 - r^2 = 0 & r &= FD \\ h_2 &= y_2^2 - 2y_1y_2 + x_2^2 - 2x_1x_2 + y_1^2 + x_1^2 - n^2 - m^2 = 0 & CD &= n^2 + m^2 \\ h_3 &= y_2^2 - 2yy_2 + x_2^2 - 2xx_2 + x^2 + y^2 - n^2 - m^2 = 0 & CB &= n^2 + m^2 \\ h_4 &= y_2^2 + x_2^2 - 2rx_2 - n^2 - 4rn - m^2 - 3r^2 = 0 & EC &= (n + 2r)^2 + m^2 \\ h_5 &= (x - r)y_1 - yx_1 + ry = 0 & E &\text{ is on } DB, \end{aligned}$$

together with the following set of pol inequations D :

$$d_1 = x_1 - x \neq 0 \quad B \neq D.$$

Selecting m , n , r , and y to be the parameters of the problem, we want to find the relation among m , n , r , y and x . Using the CS method in [Chou and Gao 1990a], a relation $x + 2n + r = 0$ is found, which tells us that the locus is a line parallel to the y -axis.

New theorems discovered in this way may be found in [Gao and Wang 1995, Wang 1992, Wu 1986a].

This problem can also be formulated as one of finding a quantifier free formula $f(u, x_1)$ such that $f(u, x_1) \iff \exists x_2 \dots x_p [h_1(u, x) \wedge \dots \wedge h_r(u, x) \wedge d_1(u, x) \neq 0 \wedge \dots \wedge d_s(u, x) \neq 0]$ [Chou 1990, Wang 1991]. Formulated in this way, it is actually to calculate the projection of an algebraic set in the affine space E^{q+p} into the subspace E^{q+1} . If E is algebraically closed, there are methods for computing such projections. The method in [Wu 1990] works over the field of complex numbers. If E is a real closed field, Collins' method [Collins 1975] gives a solution to the above problem; so it would also be interesting to examine the connection between the real closed case and algebraically closed case. For example, we expect that Collins'

method produces the following form for $f(u, x_1)$ for Peaucellier's Linkage:

$$(x + 2n + r = 0) \wedge (-d \leq y \leq d) \wedge (\text{other nondegenerate conditions})$$

where d is from the 4th (quadratic) equation of the above ASC_1^* .

2.4.2. Differential geometry and mechanics

Formula derivation in differential geometry was initiated by Wu in connection with finding possibly unknown properties on Bertrand curves [Wu 1987c]. The approach used by Wu was to look at the differential pols produced during generation of a CS. This involves human assistance. A more automatic method has been proposed in [Chou and Gao 1993c, Chou and Gao 1990a]. Using this method a complete list of the properties of Bertrand curves in metric and affine geometries has been obtained [Chou and Gao 1993c].

3. Coordinate-free approaches to automated reasoning in geometry

Algebraic methods, though powerful, generally can only tell whether a statement is true or not. If one wishes to look at the proofs, he/she will find tedious and formidable computations of pols. After Wu's method, several researchers tried to develop AGTP methods based on vector calculation in the mid-80s in order to find simpler proofs [Havel 1991, White and Mcmillan 1988]. It is well known that incidence geometry relations can be represented by exterior products and measurement geometric relations can be represented by inner products. Therefore, developing vector approach of AGTP is to find algorithms of manipulating exterior and inner products. In the mid-90s, several successful vector approaches were proposed. As expected, these methods can produce shorter proofs than that of the coordinate based methods. But, this advantage comes with a price: these methods are not complete in complex or real geometries as the methods introduced in Section 2 are.

3.1. Area method

The area method was proposed in 1992 from a quite different point of view [Chou et al. 1993a, Chou, Gao and Zhang 1994, Zhang, Chou and Gao 1995]. Zhang found many elegant ad hoc methods based on areas of triangles to solve geometric problems when he was a middle school teacher and trainer of the Chinese Mathematical Olympian Team [Zhang 1982]. These ad hoc methods have been developed into a complete method of AGTP, which are surprisingly powerful in that it has been used to prove hundreds of geometry theorems of constructive type and the proofs are generally short and elegant [Chou et al. 1994]. A computer program called *Geometry Expert* based on this method has produced proofs of 500 nontrivial theorems *entirely* automatically [Chou et al. 1994, Gao, Zhang and Chou 1998]. This method seems to be the first to produce human-readable proofs for hard geometry theorems efficiently.

Instead of coordinates, three basic *geometric quantities*: the ratio of parallel line segments, the signed area, and the Pythagorean difference are used. The basic propositions, which formally describe the properties of these quantities, are the deductive basis of the area method. The method involves the elimination of the constructed points from the conclusion using these basic geometry propositions. Two of the basic propositions are given below.

3.1. LEMMA (The Co-side Theorem). *Let M be the intersection of two lines AB and PQ and $Q \neq M$. Then $\frac{\overline{PM}}{\overline{QM}} = \frac{S_{PAB}}{S_{QAB}}$, where S_{PAB} and S_{QAB} are the signed area of triangles PAB and QAB .*

3.2. LEMMA. $PQ \parallel AB$ iff $S_{PAB} = S_{QAB}$.

We use a simple example to illustrate how the method works. The following proof is essentially the same as the one produced by the prover based on the area method.

Proof of Example 2.1. By the co-side theorem, $\frac{\overline{AE}}{\overline{EC}} = \frac{S_{ABD}}{S_{DBC}}$. Since $AB \parallel CD$, $S_{ABD} = S_{ABC}$ by Lemma 3.2. Since $AD \parallel BC$, $S_{DBC} = S_{ABC}$ by Lemma 3.2. Then we have

$$\frac{\overline{AO}}{\overline{OC}} = \frac{S_{ABD}}{S_{DBC}} = \frac{S_{ABC}}{S_{ABC}} = 1.$$

The area method has been extended to prove theorems in solid geometry (the volume method), Minkowskian geometry, Bolyai-Lobachevsky geometry, and Riemannian geometry [Chou, Gao and Zhang 1995, Yang, Gao, Chou and Zhang 1998]. In [Chou, Gao and Zhang 1996b], a new geometric quantity *the full-angle* is used to prove geometry theorems. This method, though not as powerful as the area method, can produce very elegant proofs for some difficult geometry theorems for which the area method fails to give short proofs. In [Chou et al. 1993b, Chou et al. 1994], vector and complex number approaches based on a similar idea is presented. The idea developed in the area method has been used to find locus of robotics arms [Yang et al. 1997] and to prove Newton's basic proposition [Fleuriot and Paulson 1998].

3.2. Bracket algebra methods

One of the earliest effort to develop coordinate free methods of geometric reasoning is to use techniques from the bracket algebra such as Cayley factorization [White and Mcmillan 1988]. The bracket algebra is a non-commutative algebra. There is still no decision method similar to that of the Gröbner basis. Therefore, bracket algebra can only be used to do "computer-aided geometric reasoning" at that time [(ed.) 1987].

In [Richter-Gebert 1995], an algorithm based on bracket algebra for proving projective geometry theorems was given. The basic idea is to represent geometric hypotheses and conclusions as algebraic relations and use simple algebraic computation to deduce the conclusion from the hypotheses. Many difficult theorems from projective geometry have been proved by the method. The proofs thus generated are very

short. Based on this technique, a program called *CINDERELLA* has been developed [Richter-Gebert and Kortenkamp 1999]. In [Bondyfalat et al. 1999], similar techniques are used to find unknown geometric properties raised from computer vision.

3.3. Clifford algebra methods

Clifford Algebra is a generalization of the Grassmann algebra. In [Li and Cheng 1996], techniques of Clifford algebra are combined with Wu's elimination method to prove geometry theorems. Many theorems have been proved with this approach. The key idea in [Li and Cheng 1996] is to use several rules of solving vector equations in vector level. But these rules alone are not complete. Complete methods can be achieved by substituting the vector by their coordinates and using Wu's characteristic set method. This method has also been used to formula derivation. A problem proposed by Erdős was partially solved [Li and Shi 1997].

In [Li 1998b, Wang 1998a], techniques of Clifford algebra are used to prove theorems of constructive type. This approach has the same scope and style as the vector version of the area method [Chou et al. 1993b]. This method loses a key feature of the area method: producing proofs with geometric meaning. On the other hand, it may provide a uniform treatment for Euclidean and several non-Euclidean geometries. In [Fèvre and Wang 1997, Fèvre and Wang 1998, Boy de la Tour, Fèvre and Wang 1998], rewrite rules and Clifford algebra are combined to prove theorems from both plane and solid geometries. This may enlarge the scope of the method to cover non-constructive statements and use many techniques from term re-writing to enhance the efficiency. Other approaches based on Clifford algebraic method can be found in [Yang, Zhang and Feng 1998, Fearnley-Sander 1998].

3.4. Gröbner bases methods

We have mentioned that bracket algebra and Clifford algebra are non-commutative algebras. For some non-commutative algebra, methods of generating Gröbner bases have been given. In [Stifter 1993], Gröbner bases of vector algebra involving exterior products are used to prove geometry theorems. Theoretically, inner products can also be introduced. This method is actually a combination of the vector approach and coordinates approach, because it introduces many scalar variables to represent geometric relations. In [Wang 1989], Gröbner basis of Clifford algebra is used for AGTP.

4. AI approaches to automated reasoning in geometry

Generally speaking, the algebraic approaches are decision procedures and are more powerful. The AI approaches are not decision procedures and are less powerful. Despite its "weakness", it is still worth improving the AI approach because this

may lead to techniques useful to automated reasoning in the general case. Even for automated geometry reasoning alone, AI methods have the following advantages. (1) Proofs produced by the AI method are generally easy to understand than proofs based on algebraic computations. (2) Using predicates only (no algebraic computation) makes the reaching of fixpoint possible. (3) Although algebraic methods can prove a much larger number of theorems, there still exist theorems (Example 4.2) which can be solved by the AI approaches elegantly but can not be solved with the algebraic approaches because to prove them excessively large computer memory is needed.

4.1. Gelernter's geometry machine

Geometry theorem proving on computers began in the 50s with the landmark work of Gelernter et al [Gelernter 1959, Gelernter, Hanson and Loveland 1960]. Several basic ideas of geometric reasoning such as using a numerical model, constructing auxiliary points, and generating geometric lemmas were studied in this work. Most of the other work on AI approach of geometric reasoning can be considered extensions of this work.

Gelernter's geometry machine uses a *backward chaining approach*: that is, it reasons from the conclusion to the hypotheses. Let H_1, \dots, H_r and G be the hypotheses and conclusion of a geometry statement, i.e., we need to prove

$$\forall \text{ geometric elements } [(H_1 \wedge \dots \wedge H_r) \Rightarrow G].$$

Then G is the *goal* of the proof procedure. To prove G , we search the *axiom or rule set* to find a rule of the following form

$$[(G_1 \wedge \dots \wedge G_r) \Rightarrow G].$$

Then for G to be valid, we need only to prove the *subgoals* G_1, \dots, G_r . Now we may repeat the above process for each of the subgoals until the subgoal is one of the hypotheses. In this way, we generate an *and-proof-tree* — meaning that to prove any goal in the tree, we need to prove all of its subgoals. On the other hand, there might be more than one rules that will lead to a goal. In this case, we need only to prove the subgoals generated from one of the rules. In other words, to prove a goal in the tree, we need only prove one of the branches. Therefore, in the general case, the proof process will generate an *and-or-proof-tree*.

Let us consider the following proof of Example 2.1.

The hypotheses are: $AB \parallel CD$, $AD \parallel BC$, $\text{coll}(E, A, C)$ (points E, A, C are collinear), and $\text{coll}(E, B, D)$. The conclusion or goal is $AE = EC$. The proof generated by the geometry machine is the same as the proof given at geometry textbooks. To prove $AE = EC$, we need to show $\triangle ECD \cong \triangle EAB$, which in turn follows from three subgoals: $AB = CD$, $\angle AEB = \angle CED$, and $\angle ECD = \angle EAB$. To prove $AB = CD$, we need to prove $\triangle ABC \cong \triangle CDA$ which follows from three subgoals: $AC = CA$, $\angle ACD = \angle CAB$, and $\angle CAD = \angle ACB$. $\angle ACD = \angle CAB$ follows from $AB \parallel CD$; $\angle CAD = \angle ACB$ follows from $AD \parallel BC$.

Opposite to the backward chaining, *forward chaining* reasons from the hypotheses to the conclusion. Most of the AI approaches to AGTP use backward chaining [Gelernter 1959, Anderson 1981, Coelho and Perceira 1986]. In [Nevins 1976], Nevins used a combination of forward chaining and backward chaining with emphasis on the forward chaining. In doing so, Nevins made many improvements in building a powerful geometry theorem prover. But still, most of the theorems proved by these methods are relatively easy.

Wos and his collaborators used their powerful general-purpose resolution theorem prover to experiment with proving theorems in Tarski's axioms for elementary geometry [McCharen et al. 1976]. The work was continued in [Quaife 1989] using the general purpose prover OTTER. Recently, extensive work were done in [Balbiani and del Cerro 1995, Balbiani 1995] using logic deduction techniques such as term rewriting to AGTP. In these work, some interesting but relatively easy theorems were proved. In [Fèvre 1998], the classical first-order logic and algebraic methods are combined to develop a hybrid deduction system which is used to produce proofs at different levels for better understanding.

4.2. A geometry deductive database

In [Chou et al. 1996c], the technique of deductive database [Gallaire, Minker and Nicola 1984] is used to geometric reasoning. The resulted program can be used to find the *fixpoint* for a geometric configuration, i.e., the system can find all the properties of the configuration that can be deduced using a fixed set of geometric rules. This program has been used to prove more than 150 difficult geometry theorems, and most of these theorems are beyond the scope of the previous provers based on AI approaches.

The idea of *structured deductive database* is presented to reduce the size of the database. Experiments with 150 problems show that this technique could reduce the size of the database by one thousand times.

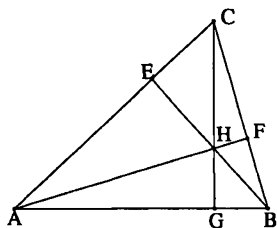


Fig. 7

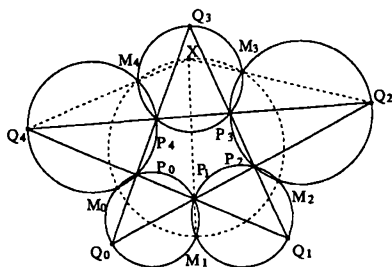


Fig. 8

4.1. EXAMPLE (*Orthocenter Theorem*). Show that the three altitudes of a triangle are concurrent (Fig. 7).

As in Fig. 7, the hypotheses (extensional database) are: points(A, B, C), coll(E, A, C), perp(B, E, A, C), coll(F, B, C), perp(A, F, B, C), coll(H, A, F),

$\text{coll}(H, B, E), \text{coll}(G, A, B), \text{coll}(G, C, H).$

Reaching the fixpoint costs the program 0.75 second on a Sparc-20. The size of the fixpoint is 151 if the structured database is used. In predicate form, the size of the fixpoint would be 83,076. The fixpoint contains two of the most often encountered properties of this configuration: $\text{perp}(C, G, A, B)$ (the conclusion of the orthocenter theorem) and $\angle[GF, GC] = \angle[GC, GE]$. For each fact in the database, the program can give a synthetic proof. The following is the proof of the Orthocenter theorem generated automatically by the program.

1. $CG \perp AB$, because $AF \perp BC$ (hypothesis), (2) $\angle[AF, BC] = \angle[CH, AB]$.
2. $\angle[AF, BC] = \angle[CH, AB]$, because (3) $\angle[AF, CH] = \angle[BC, BA]$.
3. $\angle[AF, CH] = \angle[BC, BA]$,
because (4) $\angle[AF, CH] = \angle[FE, AC]$, (5) $\angle[BC, BA] = \angle[FE, AC]$.
4. $\angle[AF, CH] = \angle[FE, AC]$, because (6) cyclic: $[C, F, E, H]$.
5. $\angle[BC, BA] = \angle[FE, AC]$, because (7) cyclic: $[A, F, B, E]$.
6. cyclic: $[C, F, E, H]$, because $FH \perp FC$ (hypothesis), $EH \perp EC$ (hypothesis).
7. cyclic: $[A, F, B, E]$, because $FB \perp FA$ (hypothesis), $EB \perp EA$ (hypothesis).

4.2. EXAMPLE. As in Fig. 8, $P_0P_1P_2P_3P_4$ is a pentagon. $Q_i = P_{i-1}P_i \cap P_{i+1}P_{i+2}$, $M_i = \text{circle}(Q_{i-1}P_{i-1}P_i) \cap \text{circle}(Q_iP_iP_{i+1})$ (the subscripts are understood to be mod 5). Show that points M_0, M_1, M_2, M_3, M_4 are cyclic.

The fixpoint contains 541 (220,680 in predicate form) facts. Besides the fact that M_0, M_1, M_2, M_3 , and M_4 are cyclic, the program finds the following new result: the following ten groups of lines

$$\{P_{i+1}M_{i+1}, Q_{i-1}M_{i-1}, Q_{i+2}M_{i-2}\}, \{P_{i-1}M_{i-2}, P_iM_{i+1}, Q_{i-1}M_{i+2}\}, i = 0, 1, 2, 3, 4$$

are concurrent and the ten intersection points of them are on the circle determined by $M_0M_1M_2$, i.e., this circle contains 15 points. The three dotted lines in Fig. 8 represent one group of concurrent lines.

4.3. Automated diagram generating

Most work on automated geometry reasoning focused on theorem proving and discovering. In [Gao and Chou 1998a], a *global propagation* method for automated generation of construction steps of diagrams was presented. This method uses a forward chaining to find the information needed in the construction and uses a backward chaining to find the construction sequences. For a diagram described declaratively with geometric constraints, the method may be used to find a sequence of constructing steps of drawing the diagram with ruler and compass.

4.3. EXAMPLE. In a solid object, there is a hollow triangle tunnel. We want to put a prism with a square cross section into the tunnel in a position as shown in Fig. 9a. We need only to consider the normal cross section. Then the problem is reduced to a plane constraint problem: to put a square into a triangle ABC (Fig. 9a).

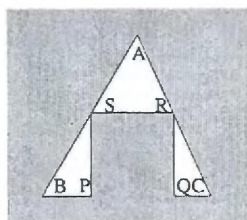


Fig. 9a

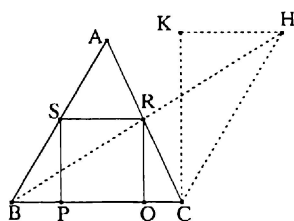


Fig. 9b

In [Gao and Chou 1998a], a solution is generated as follows (Fig. 9b): since $RQ/RS = 1$, $RQ \perp BC$, and $RS \parallel BC$, R is on a line BH , where H is the intersection of the line passing through C and parallel to AB and the line with distance $|BC|$ to line BC .

Methods of automated diagram generation have direct applications. They are the central topic in much of the current work of developing intelligent CAD systems [Brüderlin 1986, Gao and Chou 1998a, Hoffmann 1995, Kramer 1992, Owen 1991]. The main advantage of intelligent CAD system is that the resulting systems accept declarative descriptions of diagrams or mechanical designs, while for conventional CAD systems the users need to specify how to draw the diagrams.

In [Wang 1996c, Gao and Chou 1998b], the CS method is used to generate geometric diagrams automatically.

4.4. Issues in developing a prover based on AI approaches

In this section, we will discuss some of main issues in developing a powerful geometry theorem prover with the AI approaches.

Numerical Model and Generating Diagram Independent Proofs. The use of the numerical diagram as the semantic model has been the cornerstone of most of the AI approaches [Gelernter 1959, Gilmore 1970, Koedinger and Anderson 1990, Coelho and Perceira 1986]. There are two benefits the provers derive from a numerical diagram. (1) The diagram is used as a filter to reject goals not consistent with its numerical representation. (2) The numerical diagram is used to determine order relations which are necessary for the prover to find a proof. In the proof of Example 2.1 in Section 4.1, when deducing $\angle ACB = \angle ADC$ from $AB \parallel CD$ we implicitly assume that points B and D are on the opposite sides of line AC . In Gelernter's geometry machine, this fact is deduced by checking the numerical model, and a formal proof is not given.

While as a counterexample the diagram is used to control the search space successfully, the second benefit of determining order relation has some theoretical problems. Since only one or several numerical examples are checked, the provers have

the risk of proving only some special cases of the theorem. Nevins [Nevins 1976] claimed that he had got rid of this drawback by adding the ordering relations to the hypotheses of the statement. This makes the situation much clear, but still does not solve the problem. First, to prepare for the order relation, people still need to consult a diagram. Second, for some geometry theorems it may happen that the order of points in *different diagrams of the same theorem* may be different.

A key idea in AGTP is clarified by Wu in his algebraic method [Wu 1984b]. Wu observed that the validity of most geometry theorems involving equalities only is independent of the relative order positions of the points involved. Such theorems belong to *unordered geometry*. In unordered geometry, the proofs of these theorems can be very simple. However, the ordinary proofs of these theorems involve the order relation, hence are not only complicated, but also not strict.

The algebraic methods are for the unordered geometry and thus capable of producing diagram independent proofs. Among the AI approaches, the deductive database approach seems to be the only one that can produce diagram independent proofs.

Adding Auxiliary Points. Constructing new points or lines is a basic method of solving geometry problems. In logic, this corresponds to the Skolemization of the existential quantifiers [Robinson 1954]. Based on similar ideas, Reiter presented a deductive system that can generate new points [Reiter 1977]. But these ideas are not implemented. The idea of adding auxiliary point has been experimented in [Gelernter 1959, Coelho and Perceira 1986] but in a very limited sense. One of the reasons that previous AI geometry provers did not prove many difficult theorems is: without techniques of adding auxiliary points, the geometric axioms used by them can not prove most of the geometry theorems at all.

Extensive experiments on constructing auxiliary points are done in [Chou et al. 1996c]; more than thirty rules of adding auxiliary points are used. The experiments show that generating new points by Skolemization arbitrarily may easily lead to search space explosion. Strategies are used to achieve effectiveness [Chou et al. 1996c]. About forty theorems were proved by adding auxiliary points.

Multiple and Shortest Proof Generation. Since generating proofs for geometry theorems becomes very fast, we may combine search techniques and these proving methods to generate multiple proofs and in particular the shortest proofs for a geometry theorem. The experiments with the area method [Chou et al. 1996a] and the full-angle method [Chou et al. 1996b] show that by selecting control strategies properly, this approach could be successful.

The basic idea is to use rules from the area method to build a rule-based reasoning system. In other words, we "relax" the deterministic style of the original area method. Using a relaxed search strategy has two positive aspects. First, this allows the program to generate multiple proofs for the same theorem. Second, this may allow us to extend the area method to prove more theorems and to produce shorter proofs.

5. Final remarks

5.1. *Purposes of studying geometric reasoning*

Geometry has always been a model of precise reasoning. It is quite natural that it is selected as one of the first mathematical branches to be experimented with when the field of AI started in the fifties. There are other reasons leading to the extensive study of geometric reasoning. The existence of a diagram for each geometry theorem makes geometry theorem proving easy to understand by general audiences. There are a huge amount of theorems in geometry and there are always new research topics.

Besides these, are there any practical purposes to study geometric reasoning? The answer is yes.

Study of geometric reasoning has led to the invention of new concepts and new algorithms. For instance, Gelernter's work led to several important ideas in automated reasoning, such as using a model and using lemmas. As another example, Wu's method leads to the rediscovery and improvements of Ritt's work on characteristic sets which has much more applications besides geometry reasoning. Also, as pointed out in [Davis 1995], study of AGTP may lead to the reviving of the classic Euclidean geometry.

Theories of geometric reasoning may have commercial potentials. Various methods developed in automated geometry reasoning can be used to solve problems from robotics [Huang and Wu 1992, Wu 1989b, Kapur 1997, Yang et al. 1997], linkage design [Gao, Zhu and Huang 1998a], computer vision [Kapur and Mundy 1988, Gao and Cheng 1998, Wang 1998a, Bondyfalat et al. 1999], intelligent CAD [Gao and Chou 1998a, Gao and Chou 1998b], intelligent CAI [Gao, Zhu and Huang 1998b, Li and Zhang 1998], solid modeling [Wu 1993, Kapur 1997], etc.

5.2. *Further research directions*

First, we should say that theories of automated geometry reasoning are quite mature in that we can not only prove most of the geometry theorems efficiently but also produce elegant proofs for most of them. In our point of view the further research should focus on developing more general purpose techniques than proving geometry theorem alone and techniques with industrial application potential. Some possible directions are:

- In the AI setting, there is a need to develop more powerful search strategies, especially strategies used to control redundant deductions. The mechanization of other traditional proof techniques in geometry such as proving by contradiction, proving by coincidence, etc., is also very interesting.
- In the setting of coordinate-free approach, the current research focus has shifted to finding general elimination theories in the vector level and building more powerful geometric models with the Clifford algebra and other algebraic technologies [Havel 1998, Fearnley-Sander and Stokes 1998, Li 1998a]. This research

direction is still at the beginning stage.

- In the setting of coordinate approach, the focus should be on powerful elimination techniques for both complex and real number fields in order to solve difficult problems raised in the practical fields such as robotics and mechanical design.
- Automated reasoning in differential geometry, especially in the theory of surfaces, still needs more efficient algorithms and practical programs.
- The field of automated diagram generating is still quite open for development. One interesting question is that can we design a rule-based complete method for ruler and compass construction?

Acknowledgments

The work reported here was supported in part by the NSF Grant CCR-9420857 and by an Outstanding Youth Grant from Chinese NSF. We would like to thank Dr. D. Wang for valuable suggestions.

Bibliography

- ANDERSON J. [1981], Turning the search of the problem space for geometry proofs, in 'Proc. IJCAI'81, Vancouver', pp. 165–170.
- ARNON D. [1988], 'Geometric reasoning with logic and algebra', *Artificial Intelligence* **37**, 37–60.
- ARNON D. AND MIGNOTTE M. [1988], 'On mechanical quantifier elimination for elementary algebra and geometry', *J. of Symbolic Computation* **5**, 237–259.
- BALBIANI P. [1995], Equation solving in geometrical theories, in 'LNCS 968', Springer-Verlag, Berlin, pp. 31–55.
- BALBIANI P. AND DEL CERRO L. F. [1995], Affine geometry of collinearity and conditional term rewriting, in 'LNCS 909', Springer-Verlag, Berlin, pp. 196–213.
- BONDYFALAT D., MOURRAIN B. AND PAPADOPOULOU T. [1999], An application of automatic theorem proving in computer vision, in 'ADG'98, LNAI 1669', Springer-Verlag, Berlin, pp. 207–231.
- BOY DE LA TOUR T., FÈVRE S. AND WANG D. [1998], Clifford term rewriting for geometric reasoning in 3d, in 'ADG'98, LNAI 1669', Beijing, pp. 130–155.
- BRAUER R. [1948], 'A note on Hilbert's nullstellensatz', *Bull. Amer. Math. Soc.* **54**, 894–896.
- BRÜDERLIN B. [1986], Constructing three-dimensional geometric objects defined by constraints, in 'Proc. of Workshop on Interactive 3D Graphics', ACM Press, New York, pp. 111–129.
- BUCHBERGER B. [1985], Gröbner bases: An algorithmic method in polynomial ideal theory, in N. Bose, ed., 'Multidimensional Systems Theory', D. Reidel Publ., pp. 184–232.
- BUCHBERGER B., COLLINS G. AND KUTZLER B. [1995], 'Algebraic methods for geometric reasoning', *Ann. Rev. Comput. Sci.* **19**, 85–119.
- CARTAN E. [1946], *Les Systèmes Différentiels Extérieurs et leurs Applications Géométriques*, Hermann, Paris.
- CHOU S. [1984], Proving elementary geometry theorems using Wu's algorithm, in W. Bledsoe and D. Loveland, eds, 'Automated Theorem Proving: After 25 years', AMS Contemporary Mathematics Series, pp. 243–286.
- CHOU S. [1987], 'A method for mechanical derivation of formulae in elementary geometry', *J. of Automated Reasoning* **3**, 291–299.

- CHOU S. [1988], *Mechanical Geometry Theorem Proving*, D. Reidel Publishing Company, Dordrecht, Netherlands.
- CHOU S. [1990], Automated reasoning in geometry using the CS and GB methods, in 'Proc. of ISSAC'90, (Tokyo)', ACM Press, New York, pp. 255-260.
- CHOU S. AND GAO X. [1989], A collection of 120 computer solved geometry problems in mechanical formula derivation, Technical Report TR-89-22, CS Department, The Univ. of Texas at Austin.
- CHOU S. AND GAO X. [1990a], Mechanical formula derivation in elementary geometries, in 'Proc. of ISSAC'90 (Tokyo)', ACM Press, New York, pp. 265-270.
- CHOU S. AND GAO X. [1990b], Ritt-Wu's decomposition algorithm and geometry theorem proving, in 'Proc. of CADE-10', Springer-Verlag, Berlin, pp. 207-220.
- CHOU S. AND GAO X. [1991], Geometry theorems proved mechanically using Wu's method, part on differential geometry, Technical Report MM Preprints, NO. 6, MMRC, Academia Sinica.
- CHOU S. AND GAO X. [1992], Proving geometry statements of constructive type, in 'Proc. of CADE-11, LNCS 607', Springer-Verlag, Berlin, pp. 20-34.
- CHOU S. AND GAO X. [1993a], 'Automated reasoning in differential geometry and mechanics: Part i. an improved version of Ritt-Wu's decomposition algorithm', *J. of Automated Reasoning* 10, 161-172.
- CHOU S. AND GAO X. [1993b], 'Automated reasoning in differential geometry and mechanics: Part ii. mechanical theorem proving', *J. of Automated Reasoning* 10, 173-189.
- CHOU S. AND GAO X. [1993c], 'Automated reasoning in differential geometry and mechanics: Part iii. mechanical formula derivation', *IFIP Tran. on Automated Reasoning* pp. 1-12.
- CHOU S. AND GAO X. [1993d], 'Automated reasoning in differential geometry and mechanics: Part iv. Bertrand curves', *J. of Sys. Sci. and Math. Sci.* 6, 186-192.
- CHOU S. AND GAO X. [1993e], Mechanical theorem proving in Riemannian geometry using Wu's method, in W. Wu and M. Cheng, eds, 'Computer Mathematics', World Scientific, Singapore, pp. 136-158.
- CHOU S., GAO X. AND ARNON D. [1992], 'On the mechanical proof of geometry theorems involving inequalities', *Advances in Computing Research* 6, 139-181.
- CHOU S., GAO X. AND MCPHEE N. [1989], A combination of Ritt-Wu's method and Collins' method, Technical Report TR-89-28, CS Department, The Univ. of Texas at Austin.
- CHOU S., GAO X. AND ZHANG J. [1993a], Automated production of traditional proofs for constructive geometry theorems, in 'Proc. of Eighth IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, pp. 48-56.
- CHOU S., GAO X. AND ZHANG J. [1993b], Mechanical geometry theorem proving by vector calculation, in 'Proc. of ISSAC'93 (Kiev)', ACM Press, New York, pp. 284-291.
- CHOU S., GAO X. AND ZHANG J. [1994], *Machine Proofs in Geometry - Automated Production of Readable Proofs for Geometry Theorems*, World Scientific, Singapore.
- CHOU S., GAO X. AND ZHANG J. [1995], 'Automated production of traditional proofs in solid geometry', *J. of Automated Reasoning* 14, 257-291.
- CHOU S., GAO X. AND ZHANG J. [1996a], 'Automated generation of readable proofs with geometric invariants part i: Multiple and shortest proof generation', *J. of Automated Reasoning* 17, 325-347.
- CHOU S., GAO X. AND ZHANG J. [1996b], 'Automated generation of readable proofs with geometric invariants part ii: Theorem proving with full angles', *J. of Automated Reasoning* 17, 349-370.
- CHOU S., GAO X. AND ZHANG J. [1996c], 'A deductive database approach to automated geometry theorem proving and discovering', *accepted by Journal of Automated Reasoning*.
- CHOU S. AND KO H. [1986], On mechanical theorem proving in Minkowskian plane geometry, in 'Proc. of IEEE Symp. of Logic in Computer Science', IEEE Computer Society Press, pp. 187-192.

- CHOU S. AND SCHELTER W. F. [1986], 'Proving geometry theorems with rewrite rules', *J. of Automated Reasoning* **2**, 253–273.
- CHOU S. AND YANG G. [1989], 'On the algebraic formulation of certain geometry statements and mechanical geometry theorem proving', *Algorithmica* **4**, 237–262.
- COELHO H. AND PERCEIRA L. M. [1986], 'Automated reasoning in geometry theorem proving with Prolog', *J. of Automated Reasoning* **2**, 329–390.
- COLLINS G. [1975], Quantifier elimination for real closed fields by cylindrical algebraic decomposition, in 'LNCS 33', Springer-Verlag, Berlin, pp. 134–183.
- COLLINS G. AND HONG H. [1991], 'Partial CAD construction in quantifier elimination', *J. of Symbolic Computation* pp. 299–328.
- CONTI P. AND TRAVERSO C. [1995], A case of automatic theorem proving in Euclidean geometry, in 'LNCS 948', Springer-Verlag, Berlin, pp. 183–193.
- DAVIS P. [1995], 'The rise, fall and possible transfiguration of triangle geometry', *The American Mathematical Monthly* **102**, 204–214.
- DAVIS P. AND CERUTTI E. [1969], 'Formac meets Pappus', *The American Mathematical Monthly* **76**, 895–905.
- DOLZMANN A. [1998], Solving geometric problems with real quantifier elimination, in 'ADG'98, LNAI 1669', Springer-Verlag, Berlin, pp. 14–29.
- DOLZMANN A., STURM T. AND WEISPFENNING V. [1996], A new approach for automatic theorem proving in real geometry, Technical Report TR MIP-9611, to appear in JAR, Universität Passau.
- (ED.) H. C. [1987], *Computer Aided Geometric Reasoning*, Proc. INRIA Workshop, Rocquencourt, France.
- FEARNLEY-SANDER D. [1998], Plane Euclidean reasoning, in 'ADG'98, LNAI 1669', Springer-Verlag, Berlin, pp. 86–110.
- FEARNLEY-SANDER D. AND STOKES T. [1998], Area in Grassmann geometry, in D. Wang, ed., 'LNAI 1360', Springer-Verlag, Berlin, pp. 141–170.
- FERRO G. AND GALLO G. [1990], 'A procedure to prove statements in differential geometry', *J. Automated Reasoning* **6**, 203–209.
- FERRO G. AND GALLO G. [1994], 'An extension of a procedure to prove statements in differential geometry', *J. Automated Reasoning* **12**, 351–358.
- FERRO G., GALLO G. AND GENNARO R. [1998], Probabilistic verification of elementary geometry statements, in D. Wang, ed., 'LNAI 1360', Berlin, pp. 87–101.
- FÈVRE S. [1998], Integration of reasoning and algebraic calculus in geometry, in D. Wang, ed., 'ADG'96, LNAI 1360', Springer-Verlag, Berlin, pp. 218–234.
- FÈVRE S. AND WANG D. [1997], Proving geometric theorems using Clifford algebra and rewrite rules, in D. Wang, ed., 'LNAI 1421', Springer-Verlag, Berlin, pp. 17–32.
- FÈVRE S. AND WANG D. [1998], Combining algebraic computing and term-rewriting for geometry theorem proving, in 'Proc. AISC'98', Plattsburgh, USA, pp. 145–156.
- FLEURIOT J. AND PAULSON L. [1998], Proving Newton's propositio Kepleriana using geometry and nonstandard analysis in Isabelle, in 'ADG'98, LNAI 1669', Springer-Verlag, Berlin, pp. 47–66.
- GALLAIRE H., MINKER J. AND NICOLA J. M. [1984], 'Logic and databases: A deductive approach', *ACM Computing Surveys* **16**(2), 153–185.
- GAO X. [1990], 'Transcendental functions and mechanical theorem proving in elementary geometries', *J. of Automated Reasoning* **6**, 403–417.
- GAO X. AND CHENG H. [1998], On the solution classification of the "P3P" problem, in Z. Li, ed., 'Proc. of ASCM'98', Lanzhou Univ. Press, Lanzhou, pp. 185–200.
- GAO X. AND CHOU S. [1998a], 'Solving geometric constraint systems, i. a global propagation approach', *Computer Aided Design* **30**(1), 47–54.
- GAO X. AND CHOU S. [1998b], 'Solving geometric constraint systems, ii. a symbolic computational approach', *Computer Aided Design* **30**(2), 115–122.

- GAO X. AND WANG D. [1995], 'On the automatic derivation of a set of geometric formulae', *J. of Geometry* **53**, 79–88.
- GAO X., ZHANG J. AND CHOU S. [1998], *Geometry Expert*, Nine Chapters Pub., TaiPei, Taiwan (in Chinese).
- GAO X., ZHU C. C. AND HUANG Y. [1998a], Building dynamic mathematical models with geometry expert, ii. linkages, in Z. Li, ed., 'Proc. of ASCM'98', Lanzhou Univ. Press, Lanzhou, pp. 15–22.
- GAO X., ZHU C. AND HUANG Y. [1998b], Building dynamic mathematical models with geometry expert, i. geometric transformations, functions and plane curves, in W. Yang, ed., 'Proc. of ATCM'98, (Tsukuba Japan)', Springer-Verlag, Singapore, pp. 216–224.
- GELERNTER H. [1959], Realization of a geometry theorem machine, in 'Proc. Int. Conf. in Info. Process', Paris, pp. 273–282.
- GELERNTER H., HANSON J. AND LOVELAND D. [1960], Empirical explorations of the geometry-theorem proving machine, in 'Proc. West. Joint Computer Conf.', pp. 143–147.
- GIANNI P., TRAGER B. AND ZACHARIAS G. [1988], 'Gröbner bases and primary decomposition of polynomial ideals', *J. of Symbolic Computation* **6**, 149–169.
- GILMORE P. [1970], 'An examination of the geometry theorem machine', *Artificial Intelligence* **1**, 171–187.
- GUERGUEB A., MAINGUENÉ J. AND ROY M. [1998], Examples of automatic theorem proving in real geometry, in 'Proc. ISSAC'94 (Oxford)', ACM Press, New York, pp. 20–24.
- HAVEL T. [1991], 'Some examples of the use of distances as coordinates for Euclidean geometry', *J. of Symbolic Computation* **11**, 579–93.
- HAVEL T. [1998], Computational synthetic geometry with Clifford algebraic, in D. Wang, ed., 'Automated Deduction in Geometry', Springer-Verlag, Berlin, pp. 102–114.
- HILBERT D. [1971], *Foundations of Geometry*, Open Court Publishing Company, La Salla, Illinois.
- HOFFMANN C. [1995], Geometric constraint solving in R^2 and R^3 , in 'Computing in Euclidean Geometry', World Scientific, Singapore, pp. 266–298.
- HONG H., WANG D. AND WINKLER F. [1995], 'Short description of existing provers', *Ann. Math. Artif. Intell* **13**, 195–202.
- HONG J. [1986], 'Can geometry be proved by an example?', *Scientia Sinica* **29**, 824–834.
- HUANG Y. Z. AND WU W. D. [1992], Kinematic solution of a Stewart platform, in 'Proc. IWMM'92', Inter. Academic Publishers, Beijing, pp. 181–188.
- KALKBRENER M. [1995], 'A generalized Euclidean algorithm for geometry theorem proving', *Ann. of Math. and AI* **13**, 73–96.
- KAPUR D. [1986], Geometry theorem proving using Hilbert's nullstellensatz, in 'Proc. of SYMSAC'86, (Toronto)', ACM Press, New York, pp. 202–208.
- KAPUR D. [1988], 'A refutational approach to geometry theorem proving', *Artificial Intelligence* **37**, 61–93.
- KAPUR D. [1997], Automated geometric reasoning: Dixon resultants, Gröbner bases and characteristic sets, in 'Automated Deduction in Geometry', Springer-Verlag, Berlin, pp. 1–36.
- KAPUR D. AND MUNDY J. [1988], 'Wu's method and its application to perspective viewing', *Artificial Intelligence* **37**, 15–36.
- KAPUR D. AND SAXENA T. [1995], Automated geometry theorem proving using the Dixon resultants, in 'Invited Talk, IMACS-95', Albuquerque, USA.
- KAPUR D., SAXENA T. AND YANG L. [1994], Algebraic and geometric reasoning with Dixon resultants, in 'Proc. of ISSAC'94, (Oxford)', ACM Press, New York, pp. 99–107.
- KAPUR D. AND WAN H. K. [1990], Refutational proofs of geometry theorems via characteristic set computation, in 'Proc. of ISSAC'90, (Tokyo)', ACM Press, New York, pp. 277–284.
- KO H. [1988], 'Geometry theorem proving by decomposition of quasi-algebraic sets: An application of the Ritt-Wu principle', *Artificial Intelligence* **37**, 95–122.
- KO H. AND HUSSAIN M. A. [1985], Alge-prover - an algebraic geometry theorem proving software, Technical Report Technical Report, 85CRD139, General Electric Company.

- KOEDINGER K. R. AND ANDERSON J. R. [1990], 'Abstract planning and perceptual chunks: Elements of expertise in geometry', *Cognitive Science* **14**, 511–550.
- KRAMER G. [1992], *Solving Geometric Constraint Systems*, MIT Press.
- KUTZLER B. [1989], Careful algebraic translations of geometry theorems, in 'Proc. of ISSACC'89', ACM Press, New York, pp. 254–263.
- KUTZLER B. AND STIFTER S. [1986], Automated geometry theorem proving using Buchberger's algorithm, in 'Proc. of SYMSAC'86, (Toronto)', ACM Press, New York, pp. 209–214.
- LI C. AND ZHANG J. [1998], Readable machine solving in geometry and icai software msg, in 'ADG'98, LNAI 1669', Springer-Verlag, Berlin, pp. 67–85.
- LI H. [1995a], Automated reasoning with differential forms, in 'Proc. ASCM'95 (Beijing)', Scientist Press, Tokyo, pp. 29–32.
- LI H. [1998a], Some applications of Clifford algebra to geometries, in 'ADG'98, LNAI 1669', Springer-Verlag, Berlin, pp. 156–179.
- LI H. [1998b], 'Vectorial equations solving for mechanical geometry theorem proving', *J. Automated Reasoning* (accepted) .
- LI H. AND CHENG M. [1996], Proving geometric theorems with Clifford algebraic method, Technical Report MM Preprints, NO. 14, MMRC, Academia Sinica.
- LI H. AND CHENG M. [1998], 'Proving geometric theorems with Clifford in differential geometries', *J. Automated Reasoning* **21**, 1–21.
- LI H. AND SHI H. [1997], 'On erdos' ten-point problem', *Acta Mathematica Sinica, New Series* **13**(2), 221–230.
- LI Z. [1995b], 'Mechanical theorem-proving of the local theory of the surfaces', *Ann. of Math. and AI* **13**, 25–46.
- LIN D. AND LIU Z. [1992], Some results on theorem proving in finite geometry, in 'Proc. IWMM'92', Inter. Academic Pub., Beijing, pp. 222–235.
- MCCHAREN J., OVERBEEK R. AND WOS L. [1976], 'Problems and experiments for and with automated theorem-proving programs', *IEEE Trans. on Computers* **C-25**, 773–782.
- MCPHEE N., CHOU S. AND GAO X. [1994], A combination of Ritt-Wu's method and Collins' method, in 'Proc. CADE-12', Springer-Verlag, Berlin, pp. 401–415.
- MITRINOVIC D., PECARIC J. AND VOLENEC V. [1989], *Recent Advances in Geometric Inequalities*, Kluwer Academic Publishers.
- NEVINS A. [1976], 'Plane geometry theorem proving using forward chaining', *Artificial Intelligence* **6**, 1–23.
- OWEN J. [1991], Algebraic solution for geometry from dimensional constraints, in 'Proc. ACM Symp. Found. of Solid Modeling, (Austin TX)', ACM Press, New York, pp. 397–407.
- QUAIFE A. [1989], 'Automated development of Tarski geometry', *J. of Automated Reasoning* **5**, 97–118.
- REGE A. [1995], A complete and practical algorithm for geometric proving, in 'Proc. 11th Ann. Symp. Comp. Geom.', Vancouver, pp. 277–286.
- REITER R. [1977], 'A semantically guided deductive system for automatic theorem proving', *IEEE Tras. on Computers* **C-25**(4), 328–334.
- RICHTER-GEBERT J. [1995], 'Mechanical theorem proving in projective geometry', *Ann. of Math. and AI* **13**, 139–172.
- RICHTER-GEBERT J. AND KORTENKAMP U. H. [1999], *Cinderella*, Springer-Verlag, Berlin.
- RITT J. [1938], *Differential Equation from Algebraic Standpoint*, Vol. 14, AMS Colloquium Publications, New York.
- RITT J. [1950], *Differential Algebra*, AMS Colloquium Publications, New York.
- ROBINSON A. [1954], Proving a theorem (as done by man, logician, or machine), in J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning', Springer-Verlag, Berlin, pp. 74–78.
- SEIDENBERG A. [1954], 'A new decision method for elementary algebra', *Annals of Math.* **60**, 365–371.

- SEIDENBERG A. [1955], 'An elimination theory for differential algebra', *University of California Pub. in Math., New Series* **3**(2), 31–66.
- STIFTER S. [1993], Geometry theorem proving in vector spaces by means of Gröbner bases, in 'Proc. of ISSAC'93, (Kiev)', ACM Press, New York, pp. 301–310.
- STOKES T. [1990], On the Algebraic and Algorithmic Properties of Some Generalized Algebras, PhD thesis, University of Tasmania.
- TARSKI A. [1951], A decision method for elementary algebra and geometry, Technical Report Report R-109, The Rand Corporation, Santa Monica. second revised ed.
- THOMAS J. M. [1954], *Differential Systems*, Amer. Math. Soc.
- WANG D. [1988], 'Proving-by-examples method and inclusion of varieties', *Kezue Tongbao* **33**, 1121–1123.
- WANG D. [1989], 'A new theorem discovered by computer prover', *J. of Geometry* **36**, 173–182.
- WANG D. [1991], Reasoning about geometric problems using an elimination method, Technical Report Medlar 24-month deliverables, DOC, Imperial College, Univ. of London.
- WANG D. [1992], Mechanical solution of a group of space geometry problems, in 'Proc. of IWMM'92', Int. Academic Pub., Beijing, China, pp. 236–243.
- WANG D. [1993], Polynomial Equations Solving and Mechanical Geometric Theorem Proving, PhD thesis, Inst. of Sys. Sci., Academia Sinica, Beijing.
- WANG D. [1995a], 'Elimination procedures for mechanical theorem proving in geometry', *Ann. of Math. and AI* **13**, 1–24.
- WANG D. [1995b], 'A method for proving theorems in differential geometry and mechanics', *J. Univ. Comput. Sci.* **1**, 658–675.
- WANG D. [1995c], Reasoning about geometric problems using an elimination method, in J. Pfalzgraf and D. Wang, eds, 'Automated practical reasoning: Algebraic approaches', Springer-Verlag, Berlin.
- WANG D. [1996a], Algebraic factoring and geometry theorem proving, in 'LNAI 814', Springer-Verlag, Berlin.
- WANG D. [1996b], Geometry machines: From AI to SMC, in 'Proc. AISMC-3, LNCS 1138', Springer-Verlag, Berlin, pp. 213–239.
- WANG D. [1996c], Geother: A geometry theorem prover, in 'Proc. of CADE-13, (New Brunswick)', Springer-Verlag, Berlin.
- WANG D. [1998a], Clifford algebraic calculus for geometric reasoning with application to computer vision, in D. Wang, ed., 'ADG'96, LNAI 1360', Springer-Verlag, Berlin.
- WANG D. [1998b], Gröbner bases applied to geometric theorem proving and discovering, in B. Buchberger and F. Winkler, eds, 'Gröbner bases and applications', Cambridge Univ. Press, Cambridge.
- WANG D. AND GAO X. [1987], Geometry theorems proved mechanically using Wu's method, part on elementary geometries, Technical Report MM Research Preprint, No. 2, MMRC, Academia Sinica.
- WANG D. AND HU S. [1987], 'Mechanical proving system for constructible theorems in elementary geometry', *J. Sys. Sci. and Math. Scis.* **7**, 163–172.
- WANG D. AND ZHI L. [1998], Algebraic factorization applied to geometric problems, in Z. Li, ed., 'Proc. ASCM'98', Lanzhou Univ. Press, Lanzhou, China, pp. 23–36.
- WEISPFENNING V. [1994], Computational geometry problems in REDLOG, in D. Wang, ed., 'Automated Deduction in Geometry', Springer-Verlag, Berlin, pp. 386–400.
- WHITE N. L. AND MCMILLAN T. [1988], Cayley factorization, in 'Proc. of ISSAC'88', ACM Press, New York, pp. 4–8.
- WINKLER F. [1990], 'Gröbner bases in geometry theorem proving and simplest degeneracy conditions', *Math. Pannonica* **1**, 15–32.
- WINKLER F. [1992], 'Automated theorem proving in nonlinear geometry', *Issues in Robotics and Nonlinear Geometry* **6**, 183–197.

- WU T. [1995], 'On a collision problem', *Acta Math. Scientia* **15**, 32–38.
- WU W. [1978], 'On the decision problem and the mechanization of theorem proving in elementary geometry', *J. Sys. Sci. and Math. Scis.* **21**, 157–179.
- WU W. [1979], 'Mechanical theorem proving in elementary differential geometry', *Scientia Sinica* pp. 94–102.
- WU W. [1982a], Mechanical theorem proving in elementary geometry and differential geometry, in 'Proc. 1980 Beijing Symposium on Differential Geometry and Differential Equations', Vol. 2, Science Press, Beijing, pp. 125–138.
- WU W. [1982b], 'Toward mechanization of geometry — some comments on Hilbert's Grundlagen der Geometrie', *Acta Math. Sci.* pp. 125–138.
- WU W. [1984a], 'Basic principles of mechanical theorem proving in geometries', *J. Sys. Sci. and Math. Scis.* **4**(3), 207–235.
- WU W. [1984b], *Basic Principles of Mechanical Theorem Proving in Geometries (in Chinese)*, Science Press, Beijing.
- WU W. [1984c], Some recent advances in mechanical theorem-proving of geometries, in 'Automated Theorem Proving: After 25 years', AMS, Contemporary Mathematics, pp. 235–242.
- WU W. [1986a], 'A mechanization method of geometry and its applications i: Distances, areas, and volumes', *J. Sys. Sci. and Math. Scis.* **6**, 204–216.
- WU W. [1986b], 'On reducibility problem in mechanical theorem proving of elementary geometries', *Chinese Quart. J. Math.* pp. 1–20.
- WU W. [1987a], A constructive theory of differential algebraic geometry, in 'Lecture Notes in Math.', Vol. 1255, Springer-Verlag, Berlin, pp. 173–189.
- WU W. [1987b], Mechanical derivation of Newton's gravitational laws from Kepler's laws, Technical Report MM Research Preprints No.1, MMRC, Academia Sinica.
- WU W. [1987c], 'A mechanization method of geometry and its applications ii: Curve pairs of Bertrand type', *Kexue Tongbao* **32**(9), 535–538.
- WU W. [1989a], 'A mechanization method of geometry and its applications iv. some theorems in planar kinematics', *J. Sys. Scis and Math. Sci.* **2**, 97–109.
- WU W. [1989b], A mechanization method of geometry and its applications vi. solving inverse kinematics equations of PUMA-type robotics, Technical Report MM Research Preprints No.4, MMRC, Academia Sinica.
- WU W. [1989c], 'On the foundation of algebraic differential geometry', *Sys. Sci. and Math. Sci.* **2**, 289–312.
- WU W. [1990], 'On a projection theorem of quasi-varieties in elimination theory', *Chinese Ann. of Math.* **11B**, 220–226.
- WU W. [1992a], On a finiteness theorem about optimization problems, Technical Report MM Research Preprints No.8, MMRC, Academia Sinica.
- WU W. [1992b], 'A report on mechanical geometry theorem proving', *Progress in Natural Sciences* **2**, 1–17.
- WU W. [1993], On surface-fitting problem in CAGD, Technical Report MM Research Preprints No.10, MMRC, Academia Sinica.
- YANG H., ZHANG S. AND FENG G. [1998], A Clifford algebraic method for geometric reasoning, in 'ADG'98, LNAI 1669', Springer-Verlag, Berlin, pp. 111–129.
- YANG L. [1998], Practical automated reasoning on inequalities, in 'Proc. of ATCM, (Tsukuba, Japan)', Springer-Verlag, Berlin, pp. 24–25.
- YANG L., FU H. AND ZHENG Z. [1997], A practical symbolic algorithm for inverse kinematics of 6R manipulators with simple geometry, in 'Proc. of CADE-14, (Townsville, Australia)', Springer-Verlag, Berlin, pp. 73–88.
- YANG L., GAO X., CHOU S. AND ZHANG J. [1998], Automated proving and discovering of theorems in non-Euclidean geometries, in D. Wang, ed., 'Automated Deduction in Geometry', Springer-Verlag, Berlin, pp. 171–188.

- YANG L., HOU X. AND XIA B. [1998], Automated discovering and proving for geometric inequalities, in 'ADG'98, LNAI 1669', Springer-Verlag, Berlin, pp. 30–46.
- YANG L., HOU X. AND ZENG Z. [1996], 'A complete discrimination system for polynomials', *Sciences in China E* **39**, 628–646.
- YANG L., ZHANG J. AND HOU X. [1992], A criterion of dependency between algebraic equations and its applications, in 'Proc. IWMM'92', Inter. Academic Publishers, Beijing, pp. 110–134.
- YANG L., ZHANG J. AND HOU X. [1996], *Non-linear Algebraic Equations and Automated Theorem Proving*, ShangHai Science and Education Pub., ShangHai.
- ZHANG J. [1982], *How to Solve Geometry Problems Using Areas*, Shanghai Educational Publishing Inc., ShangHai.
- ZHANG J., CHOU S. AND GAO X. [1995], 'Automated production of traditional proofs for theorems in Euclidean geometry', *Annals of Math. and AI* **13**, 109–137.
- ZHANG J., YANG L. AND DENG M. [1990], 'The parallel numerical method of mechanical theorem proving', *Theoretical Computer Science* **74**, 253–271.

Index

A

area method732
 automated diagram generating 737
 automated formula derivation 730
 auxiliary point 739

B

backward chaining735
 bracket algebra method733
 Brauer-Seidenberg-Wang method 722

C

characteristic set method 712, 719
 Clifford algebra method 734
 Collins' CAD algorithm 724
 complex number method 733

D

deductive database method736
 Dixon resultant method 722

F

fixpoint 736
 forward chaining 735
 full-angle method733

G

Gelernter's geometry machine 735
 generically false 718
 generically true718
 geometry statement 712
 global propagation 737
 Gröbner bases methods734
 Gröbner basis method 719

H

Hilbert's method 715

I

intelligent CAD 738

K

Kepler-Newton problem728

L

logic approaches736

M

multiple proof generation 739

N

numerical model738

P

parallelogram theorem712
 Peaucellier's linkage 731
 proving inequalities724
 proving theorems by checking examples 723
 proving theorems in differential geometry 728
 pseudo division713

R

resultant method721

S

shortest proof generation 739
 Simson's Theorem715
 structured deductive database 736
 subsidiary conditions 714

T

triangular set714

V

vector method733

W

Wu's method 715, 719

Solving Numerical Constraints

Alexander Bockmayr

Volker Weispfenning

SECOND READERS: Michael Maher.

Contents

1	Introduction	753
1.1	Constraints in automated reasoning	753
1.2	A classification of numerical constraints	754
1.3	Preliminaries	755
2	Linear constraints over fields	758
2.1	Linear equations	758
2.2	Linear inequalities	761
2.3	Disequalities	774
2.4	Extensions	775
3	Linear diophantine constraints	779
3.1	Equations over \mathbb{Z}	779
3.2	Equations over \mathbb{N} , inequalities over \mathbb{Z}	782
3.3	Cutting planes	785
3.4	Branch-and-infer	788
3.5	Gröbner bases in integer programming	792
3.6	Computing Hilbert bases	793
3.7	Linear 0-1 inequalities	798
3.8	Presburger arithmetic	800
4	Non-linear constraints over continuous domains	802
4.1	Zero-sets of univariate polynomials	802
4.2	Zero-sets of multivariate polynomials	805
4.3	Parametric zero-sets	812
4.4	Complex and real quantifier elimination	816
4.5	Further Topics	818
5	Non-linear diophantine constraints	819
5.1	Hilbert's Tenth Problem	819
5.2	Positive results	821
5.3	Decision problems for first-order theories of the integers	822
	Bibliography	823
	Index	838

HANDBOOK OF AUTOMATED REASONING

Edited by Alan Robinson and Andrei Voronkov

© 2001 Elsevier Science Publishers B.V. All rights reserved

1. Introduction

1.1. Constraints in automated reasoning

Numerical constraints have been at the heart of mathematics for about 4000 years. By the second millennium B.C., the Babylonians and Egyptians solved simple examples of linear equations. The Babylonians, Greeks, and Chinese knew the idea of elimination of unknowns for both linear and quadratic equations. Over the course of history, many fields of mathematics have contributed to the study of numerical constraints, e.g., algebra, geometry, number theory, analysis, and optimization theory. During the last 40 years, numerical constraints have also become a major topic of interest in computer science. This paper considers numerical constraint solving in the context of automated reasoning. We are mainly interested in logical aspects of numerical constraints. We view them as atomic formulae in first-order predicate logic that are interpreted over some numerical domain like the real, rational, or integer numbers. We will present various inference systems for reasoning with numerical constraints, describe the algebraic and geometric structure of their solution set, and present fundamental algorithms for deciding satisfiability, computing one or all solutions, and deducing new constraints. We will focus on exact symbolic methods, i.e. we will not consider approximative algorithms or interval techniques that are normally studied in numerical analysis. We will also not discuss numerical problems like rounding errors.

Enhancing general theorem proving by specialized procedures for a particular domain has been a pertinent research theme in automated reasoning, see e.g. [Huet 1972, Plotkin 1972, Stickel 1985, Bürckert 1991] or for more recent work [Björner, Stickel and Uribe 1997, Armando and Ranise 1998, Bertoli, Calmet, Giunchiglia and Homann 1998, Dowek, Hardin and Kirchner 1998, Janičić, Bundy and Green 1999]. Cooper [1972] studied theorem proving in Presburger arithmetic. Linear arithmetic over the rational numbers was investigated, e.g., in [Bledsoe 1975, Shostak 1977, Shostak 1979, Boyer and Moore 1988, Käuff 1988, Kapur and Nie 1994]. Methods for the combination of decision procedures for different theories were developed by Nelson and Oppen [1979] and Shostak [1984]. The notion of deduction with constraints can be traced back at least to [Huet 1972]. It became very popular through the work on constraint logic programming [Colmerauer 1987, Jaffar and Lassez 1987]. A framework for constrained deduction was developed in [Kirchner, Kirchner and Rusinowitch 1990], see also [Kirchner 1995]. A methodological view of constraint solving is presented in [Jouannaud and Kirchner 1991, Comon 1991, Comon, Dincbas, Jouannaud and Kirchner 1999]. Armando, Melis and Ranise [1998] compare constraint solving in logic programming and automated deduction. For more information on the integration of constraint solving into different theorem proving methods, we refer to the [Bachmair and Ganzinger 2001, Nieuwenhuis and Rubio 2001, Degtyarev and Voronkov 2001a], i.e., Chapters 2, 7, and 10 in this handbook. The special case of unification is discussed in [Baader and Snyder 2001] (Chapter 8).

The field that we are trying to cover in this chapter is huge. We can only give

Satisfiability	over \mathbb{R}	over \mathbb{Z}
Linear constraints	polynomial	NP-complete
Non-linear constraints	decidable	undecidable

Table 1: Solving numerical constraints

an outline of what we think are the most important issues. Each section or even subsection of this chapter would deserve a paper or book on its own. We have included an extensive bibliography for further reference. We invite the reader to consult some of the monographs and articles listed there in order to learn more about this fascinating field.

1.2. A classification of numerical constraints

Numerical domains are the field \mathbb{C} of complex numbers, the field \mathbb{R} of real numbers, the field \mathbb{Q} of rational numbers, the ring \mathbb{Z} of integer numbers, the semiring \mathbb{N} of natural numbers (including zero), and the two-element set $\{0, 1\}$. We will use the symbol \mathbb{D} to denote a numerical domain, and the symbol \mathbb{F} to denote one of the fields \mathbb{C}, \mathbb{R} , or \mathbb{Q} . A *numerical function* is a mapping $f : \mathbb{D}_1 \times \cdots \times \mathbb{D}_n \rightarrow \mathbb{D}$ from a Cartesian product of numerical domains $\mathbb{D}_1 \times \cdots \times \mathbb{D}_n$ into a numerical domain \mathbb{D} . We are mainly interested in *linear functions*

$$f(x_1, \dots, x_n) = \sum_{i=1}^n c_i x_i, \quad c_i \in \mathbb{D}, \tag{1.1}$$

and *polynomials*

$$p(x_1, \dots, x_n) = \sum_{\substack{(i_1, \dots, i_n) \in \mathbb{N}^n \\ i_1 + \dots + i_n \leq k}} c_{i_1, \dots, i_n} x_1^{i_1} \dots x_n^{i_n}, \quad c_{i_1, \dots, i_n} \in \mathbb{D}, k \in \mathbb{N}. \tag{1.2}$$

A *numerical constraint* is an equation, inequality, or disequality between two numerical functions. We exclude inequalities between complex-valued functions. By $\text{Sol}_{\mathbb{D}}(C)$ we denote the set of solutions of the constraint system C in the domain \mathbb{D} . We are mainly interested in conjunctions of atomic formulae, but we will also discuss arbitrary Boolean combinations and quantifier elimination in the corresponding first-order theories.

We classify constraints according to whether the numerical functions are linear or non-linear, and whether they are solved over a continuous domain, i.e. \mathbb{R}, \mathbb{C} , or a discrete domain, i.e. \mathbb{Z}, \mathbb{N} or $\{0, 1\}$. Very roughly, we get the picture represented in Table 1. The rational numbers \mathbb{Q} play a special role and are not included in this table. In the linear case, they behave like the reals, in the non-linear case more like a discrete domain. In order to get a more accurate picture in the linear case, we have to distinguish between equations and inequalities, see Table 2.

Satisfiability	over \mathbb{Q}	over \mathbb{Z}	over \mathbb{N}
Linear equations	polynomial	polynomial	NP-complete
Linear inequalities	polynomial	NP-complete	NP-complete

Table 2: Solving linear constraints

1.3. Preliminaries

We start with some preliminaries from linear algebra, complexity theory, commutative algebra, and first-order logic. Basic references are [Schrijver 1986, Grötschel, Lovász and Schrijver 1988, Becker, Weispfenning and Kredel 1993, von zur Gathen and Gerhard 1999].

1.3.1. Linear algebra

For $n \in \mathbb{N}, n \geq 1$, we denote by \mathbb{D}^n the set of vectors with n components and entries in \mathbb{D} . Addition of vectors and multiplication of vectors with scalars are defined as usual. A vector is always considered as a column vector. The superscript \cdot^T denotes transposition, i.e. for $x \in \mathbb{D}^n$, x^T is a row vector.

For $m, n \in \mathbb{N}, m, n \geq 1$, we denote by $\mathbb{D}^{m \times n}$ the set of $m \times n$ -matrices with entries in \mathbb{D} . For a matrix $A \in \mathbb{D}^{m \times n}$, we usually assume that the row index set of A is $M = \{1, \dots, m\}$ and that the column index set is $N = \{1, \dots, n\}$. For $I \subseteq M, J \subseteq N$, we denote by A_{IJ} the submatrix of A induced by the rows in I and the columns in J . Instead of A_{MJ} and A_{IN} , we will also write A_{*J} and A_{I*} . A_{i*} is the i -th row of A , and A_{*j} the j -th column. Unless specified otherwise, the elements of $A \in \mathbb{D}^{m \times n}$ are denoted by $a_{ij}, 1 \leq i \leq m, 1 \leq j \leq n$. Vectors with n components are also considered as $n \times 1$ -matrices. Given $A \in \mathbb{D}^{m \times l}$ and $B \in \mathbb{D}^{l \times n}$, the product $C = A \cdot B \in \mathbb{D}^{m \times n}$ is defined by $c_{ij} = \sum_{k=1}^l a_{ik} \cdot b_{kj}$.

A vector $x \in \mathbb{R}^n$ is called a *linear combination* of the vectors $x_1, \dots, x_k \in \mathbb{R}^n$ if $x = \lambda_1 x_1 + \dots + \lambda_k x_k$, for some $\lambda_1, \dots, \lambda_k \in \mathbb{R}$. If, in addition

$$\left\{ \begin{array}{l} \lambda_1, \dots, \lambda_k \geq 0, \\ \lambda_1 + \dots + \lambda_k = 1, \end{array} \right\} \quad \text{we call } x \text{ a } \left\{ \begin{array}{l} \text{conic} \\ \text{affine} \\ \text{convex} \end{array} \right\} \text{ combination.}$$

These combinations are *proper* if neither $\lambda_1 = \dots = \lambda_k = 0$ nor $x = x_i$, for some $i = 1, \dots, k$. For a nonempty subset $S \subseteq \mathbb{R}^n$, we denote by $\text{lin}(S)$ (resp. $\text{cone}(S)$, $\text{aff}(S)$, $\text{conv}(S)$) the *linear* (resp. *conic*, *affine*, *convex*) *hull* of S , i.e. the set of all linear (resp. conic, affine, convex) combinations of finitely many vectors of S . For the empty set, we define $\text{lin}(\emptyset) = \text{cone}(\emptyset) = \{0\}$ and $\text{aff}(\emptyset) = \text{conv}(\emptyset) = \emptyset$. A subset $S \subseteq \mathbb{R}^n$ is called a *linear subspace* (resp. *cone*, *affine subspace*, *convex set*) if $S = \text{lin}(S)$ (resp. $S = \text{cone}(S)$, $S = \text{aff}(S)$, $S = \text{conv}(S)$).

A subset $S \subseteq \mathbb{R}^n$ is called *linearly* (resp. *affinely*) *independent* if none of its members is a proper linear (resp. affine) combination of elements of S . The *dimen-*

sion $\dim(P)$ of a subset $P \subseteq \mathbb{R}^n$ is the maximum number of affinely independent vectors in P minus one. A set $P \subseteq \mathbb{R}^n$ is *full-dimensional* if $\dim(P) = n$. The *rank* of an $m \times n$ -matrix A is the maximal number of linearly independent columns (or equivalently rows) of A . The matrix A has *full row rank* (resp. *full column rank*) if $\text{rank}(A) = m$ (resp. $\text{rank}(A) = n$).

1.3.2. Complexity

The *encoding length* of rational numbers, vectors, and matrices is defined as follows:

$$\begin{aligned} \langle n \rangle &= 1 + \lceil \log_2(|n| + 1) \rceil, & n &\in \mathbb{Z} \\ \langle r \rangle &= \langle p \rangle + \langle q \rangle, & r &= p/q \in \mathbb{Q}, q > 0, \gcd(p, q) = 1 \\ \langle a \rangle &= \langle a_1 \rangle + \cdots + \langle a_n \rangle, & a &= (a_1, \dots, a_n)^T \in \mathbb{Q}^n \\ \langle A \rangle &= \langle a_{11} \rangle + \cdots + \langle a_{mn} \rangle, & A &= (a_{ij}) \in \mathbb{Q}^{m \times n} \end{aligned}$$

Strongly polynomial algorithms. When analyzing numerical algorithms it is often natural to use the *arithmetic model* of computation, where we count the number of elementary arithmetic operations, i.e. addition, subtraction, multiplication, division, and comparison, instead of the number of steps of a Turing machine. In the arithmetic model, the *encoding length* of a problem instance is simply the number of input numbers, not the encoding length of these numbers.

The running time of an algorithm may be polynomially bounded in the Turing machine model, but not in the arithmetic model, and vice versa. For example, the Euclidean algorithm to compute the greatest common divisor of two integers is polynomial in the Turing machine model, but not in the arithmetic model. Conversely, a polynomial number of elementary arithmetic operations cannot be executed in polynomial time on a Turing machine if the encoding length of the numbers occurring in the computation is not polynomially bounded. We say that an algorithm is *strongly polynomial* if it is a polynomial space algorithm in the Turing machine model and also a polynomial time algorithm in the arithmetic model. This definition depends on how we encode rational numbers and the result of arithmetic operations. We will assume that if a and b are two integers such that we know in advance that b divides a , then the rational number a/b will be represented by the corresponding integer [Grötschel et al. 1988, p.32-33].

1.3.3. Polynomials

The polynomials in an indeterminate x with coefficients in the field \mathbb{F} form a commutative ring, the *univariate polynomial ring* $\mathbb{F}[x]$. The polynomials in several indeterminates x_1, \dots, x_n with coefficients in the field \mathbb{F} form another commutative ring, the *multivariate polynomial ring* $\mathbb{F}[x_1, \dots, x_n]$. For an extension ring S of \mathbb{F} , and elements c_1, \dots, c_n in S , the simultaneous substitution of c_i for x_i is compatible with the ring operations. This important fact will be used tacitly whenever needed. As usual, the result of this substitution for a polynomial $f = f(x_1, \dots, x_n)$ will be denoted by $f(c_1, \dots, c_n) \in S$. The n -tuple (c_1, \dots, c_n) is a *zero* of f if $f(c_1, \dots, c_n) = 0$. The set of all common zeros of a set $G \subseteq \mathbb{F}[x_1, \dots, x_n]$ in \mathbb{F}^n

is called the \mathbb{F} -variety of G and is denoted by $V_{\mathbb{F}}(G)$. A multivariate polynomial f is *homogeneous* if all monomials occurring in f with non-vanishing coefficient have the same degree. Homogeneous polynomials always have the origin as trivial zero; every other zero is called a *non-trivial zero*. An extension field \mathbb{K} of \mathbb{F} is *algebraically closed*, if every non-constant univariate polynomial $f \in \mathbb{F}[x]$ has a zero in \mathbb{K} . In particular, the field \mathbb{C} of complex numbers is algebraically closed. An ordered extension field \mathbb{K} of \mathbb{F} is *real-closed*, if the intermediate value theorem of analysis holds for all polynomial functions on \mathbb{K} . In particular, the field \mathbb{R} of real numbers is real-closed.

1.3.4. First-order theories

The basic concepts of first-order logic that we require are as follows. For a given numerical domain such as $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ we specify the constants, operations, and relations to be considered. A formal list of symbols for these objects together with the arities of the operation symbols and relation symbols forms a *first-order language* L . With every such language we associate a countably infinite set of symbols called *variables*, and some auxiliary symbols such as parentheses and commas. Among all finite strings of L -symbols we distinguish in particular the L -terms and L -formulas: L -terms are formal superpositions of operation symbols starting from variables and constants. In the following we consider only languages L having only binary relations symbols. Then *atomic* L -formulas are strings of the form $s \rho t$, where s, t are L -terms and ρ is a relation symbol in L . L -formulas are obtained from atomic L -formulas by composition with the propositional operators \wedge (and), \vee (or), \neg (not), and quantifiers $\exists x, \forall x$ for variables x with appropriate use of commas and parentheses. An occurrence of a variable x in the scope of a corresponding quantifier $\exists x, \forall x$ is called *bound*. All other occurrences of a variable in an L -formula are *free*. An L -formula is *quantifier-free*, if it contains no quantifiers. An L -formula φ is *prenex*, if it is of the form $Q_1 x_1 \dots Q_n x_n \psi$, where ψ is a quantifier-free formula and Q_i are quantifiers. The part $Q_1 x_1 \dots Q_n x_n$ is called the *prefix* of the formula φ . In this prefix the quantifiers $Q_i x_i$ can be grouped into *blocks* of similar quantifiers (i.e. all existential or all universal). An *existential formula* is a prenex formula with a single block of existential quantifiers. Similarly, a *universal formula* is a prenex formula with a single block of universal quantifiers. A *closed* L -formula is an L -formula in which all variables x are within the scope of a matching quantifier $\exists x$ or $\forall x$. In other words, a closed L -formula contains no *parameters*.

Every closed L -formula has a natural meaning in a numerical domain \mathbb{D} where the constants, operations and relations named in L are defined. So it assumes exactly one of the truth values 'true' or 'false'. For arbitrary L -formulas the meaning depends on an assignment of numerical values in \mathbb{D} for all free occurrences of variables. Two L -formulas are *equivalent* in \mathbb{D} if they assume the same truth value for every variable assignment in \mathbb{D} . A basic fact of first-order logic asserts that every L -formula is equivalent over an arbitrary domain to a prenex L -formula. Moreover any quantifier-free L -formula is equivalent to an L -formula in *disjunctive normal form*, i.e. to a disjunction of conjunctions of atomic L -formulas and negations of

atomic L -formulas. We say that an arbitrary formula holds in \mathbb{D} if it assumes the truth value ‘true’ for every variable assignment in \mathbb{D} . The *first-order L -theory* of some domain \mathbb{D} is the set of all L -formulas that hold in \mathbb{D} .

1.3.5. Quantifier elimination and decision procedures

A *quantifier elimination* procedure for a theory T of some domain \mathbb{D} is an algorithm that assigns to every L -formula φ a quantifier-free L -formula ψ that is equivalent to φ in \mathbb{D} . A *decision procedure* for T is an algorithm that decides for a given closed formula whether or not this formula holds in the domain \mathbb{D} . From a given quantifier elimination procedure for T for a domain \mathbb{D} , where the truth value of closed atomic L -formulas can be evaluated algorithmically, it is easy to produce a decision procedure: on input of a closed formula φ , the quantifier elimination procedure computes an equivalent closed quantifier-free and hence variable-free formula ψ that is a Boolean combination of relations between constant terms. So the truth value of ψ can be determined by evaluating all atomic parts of ψ . While the decision problem is concerned with closed (i.e. parameter-free) formulas, the quantifier elimination problem deals with simplification of formulas involving parameters to a form that is easy to evaluate. For applications, a quantifier elimination procedure is much more useful than a decision procedure. The reason is that a decision procedure gives only a ‘yes’ or ‘no’ answer to a specific problem, while a quantifier elimination procedure describes for which numerical values of the parameters in \mathbb{D} an assertion holds; thus quantifier elimination provides a structural explanation in place of a simple yes/no-answer.

2. Linear constraints over fields

We begin our discussion with classical results from linear algebra and linear optimization. The presentation in Sections 2.1 to 2.2.3 is based on [Schrijver 1986, Grötschel et al. 1988, Bertsimas and Tsitsiklis 1997], where the reader may find many additional details and the proofs of the main theorems. First we study linear equations over a field \mathbb{F} .

2.1. Linear equations

2.1.1. Basic theory

Consider a system of linear equations

$$\begin{array}{rcl} a_1^T x & = & \beta_1 \\ \vdots & & \vdots \\ a_m^T x & = & \beta_m \end{array} \iff Ax = b, \quad (2.1)$$

with $a_1, \dots, a_m \in \mathbb{F}^n$, $\beta_1, \dots, \beta_m \in \mathbb{F}$, $A \in \mathbb{F}^{m \times n}$, and $b \in \mathbb{F}^m$. The basic inference principle to derive a new equation $c^T x = \delta$, with $c \in \mathbb{F}^n$, $\delta \in \mathbb{F}$, is taking a *linear*

combination of the given equations. More precisely, we can multiply each equation $a_i^T x = \beta_i$ by a number $u_i \in \mathbb{F}$ and add up the resulting equations $u_i a_i^T x = u_i \beta_i$, for $i = 1, \dots, m$. In matrix notation, this can be expressed by the inference rule

$$\text{lin_com} : \frac{Ax = b}{(u^T A)x = u^T b} \text{ if } u \in \mathbb{F}^m. \quad (2.2)$$

If $Ax = b$ is solvable in \mathbb{F}^n , the rule `lin_com` allows us to derive any implied equation $c^T x = \delta$. If $Ax = b$ is not solvable, we can get the contradiction $0 = 1$.

2.1. PROPOSITION (Solvability and entailment). *The system $Ax = b$ is not solvable in \mathbb{F}^n if and only if there exists $u \in \mathbb{F}^m$ such that $u^T A = 0$ and $u^T b = 1$. If $Ax = b$ is solvable, an equation $c^T x = \delta$ is implied by $Ax = b$ in \mathbb{F}^n if and only if there exists $u \in \mathbb{F}^m$ such that $u^T A = c^T$ and $u^T b = \delta$.*

The solution set of a single linear equation $a^T x = \beta$ defines a *hyperplane* in the n -dimensional affine space \mathbb{F}^n . The solution set S of a system of linear equations $Ax = b$ corresponds to an intersection of hyperplanes, i.e. an *affine subspace*. For $x^0 \in S$ the set $U = \{x - x^0 \mid x \in S\}$ defines a linear subspace of the n -dimensional vector space \mathbb{F}^n . If $\dim U = k$, then U is generated by k linearly independent vectors $x^1, \dots, x^k \in \mathbb{F}^n$.

2.2. PROPOSITION (Structure of the solution set). *If the system $Ax = b$ of linear equations has a solution $x^0 \in \mathbb{F}^n$, then there exist vectors $x^1, \dots, x^k \in \mathbb{F}^n$ such that*

$$\text{Sol}_{\mathbb{F}}(Ax = b) = x^0 + \mathbb{F} \cdot x^1 + \dots + \mathbb{F} \cdot x^k. \quad (2.3)$$

2.1.2. Gaussian elimination

Gaussian elimination is probably the best-known algorithm in linear algebra. To solve a system of linear equations

$$\begin{array}{ccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \dots & + & a_{mn}x_n & = & b_m, \end{array} \quad (2.4)$$

the basic idea is to eliminate successively the variables in the problem. Suppose $a_{11} \neq 0$. Then we can subtract appropriate multiples of the first equation from the other equations, which corresponds to applications of the rule `lin_com`, in order to get

$$\begin{array}{ccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ & & a'_{22}x_2 & + & \dots & + & a'_{2n}x_n & = & b'_2 \\ & & \vdots & & & & \vdots & & \vdots \\ & & a'_{m2}x_2 & + & \dots & + & a'_{mn}x_n & = & b'_m. \end{array} \quad (2.5)$$

Now we solve recursively the system consisting of the last $m - 1$ equations. Substituting the solution into the first equation yields a value for x_1 .

Forward and backward phase. In matrix form, Gaussian elimination transforms a matrix $A \in \mathbb{F}^{m \times n}$ into a new matrix of the form

$$\begin{pmatrix} \Delta & D \\ 0 & 0 \end{pmatrix}, \quad (2.6)$$

where Δ is a diagonal $r \times r$ -matrix and r is the rank of A . In the *forward phase*, we compute matrices A_0, A_1, \dots, A_r , where A_k is of the form

$$A_k = \begin{pmatrix} B & C \\ 0 & D \end{pmatrix}, \quad (2.7)$$

with a non-singular upper triangular matrix $B \in \mathbb{F}^{k \times k}$, for $k = 0, \dots, r$. We start with $A_0 = A$. Given A_k of form (2.7), we choose a non-zero element of D , called the *pivot element*, and permute the rows and columns of A , so that this element becomes d_{11} . Now we add multiples of the first row of D to the other rows of D such that d_{11} becomes the only nonzero element in the first column of D , i.e.

$$d_{ij} := d_{ij} - \frac{d_{i1}}{d_{11}} d_{1j}, \text{ for } i = 1, \dots, m - k, j = 1, \dots, n - k. \quad (2.8)$$

The resulting matrix is called A_{k+1} . This is repeated until D contains no nonzero element. In the *backward phase*, we start with the matrix $E_r = A_r$ and compute matrices E_{r-1}, \dots, E_0 of the form

$$E_k = \begin{pmatrix} B & 0 & C \\ 0 & \Delta & D \\ 0 & 0 & 0 \end{pmatrix}, \quad (2.9)$$

such that B is a non-singular upper triangular $k \times k$ -matrix and Δ is a diagonal $(r - k) \times (r - k)$ -matrix. Given the matrix E_k , we add multiples of the k -th row to the other rows of E_k so that b_{kk} will be the only nonzero entry in the k -column of E_k . The resulting matrix is E_{k-1} . The final matrix E_0 has the form (2.6).

Complexity. Obviously, the number of elementary arithmetic operations (addition, subtraction, multiplication, division, comparison) in Gaussian elimination is bounded by a polynomial in m and n . If we assume $n = m$, then we need $O(n^2)$ divisions and $O(n^3)$ additions and multiplications. Note that this is not the best possible. Starting with the classical work by Strassen [1969], fast matrix multiplication algorithms have been developed that require less than $O(n^3)$ arithmetic operations. The best currently known value is less than $O(n^{2.376})$ [Coppersmith and Winograd 1990, Bürgisser, Clausen and Shokrollahi 1997, von zur Gathen and Gerhard 1999].

To show that Gaussian elimination is polynomial, we have to prove in addition that the encoding length of the numbers produced by the algorithm is polynomially

bounded. This can be guaranteed if rational numbers are represented in *coprime* form, i.e. by fractions p/q with $\gcd(p, q) = 1$.

2.3. THEOREM (Edmonds 67). *For any rational matrix $A \in \mathbb{Q}^{m \times n}$, the Gaussian elimination method (using coprime representation of rationals) runs in polynomial time.*

There exists a representation scheme for rational numbers such that the Gaussian elimination method runs even in strongly polynomial time [Edmonds 1967, Bareiss 1968, Grötschel et al. 1988, Geddes, Czapor and Labahn 1992]. This implies:

2.4. COROLLARY. *There exist strongly polynomial algorithms for the following problems:*

- *Finding a solution of a rational equation system $Ax = b$,*
- *Determining the rank of a rational matrix,*
- *Determining the determinant of a rational matrix,*
- *Determining the inverse of a non-singular rational matrix,*
- *Testing whether m given rational vectors are linearly independent.*

Gauss-Jordan elimination is another polynomial time method that combines forward and backward steps. In the k -th step, we have a matrix of the form

$$A_k = \begin{pmatrix} I & C \\ 0 & D \end{pmatrix}, \quad (2.10)$$

where I is the $k \times k$ -identity matrix. Again, we choose a non-zero pivot element in D , and permute rows and columns so that this element becomes d_{11} . Then we divide the $(k+1)$ -th row of A_k by d_{11} , and add rational multiples of this row to all the other rows, such that the 1 in position $(k+1, k+1)$ is the only non-zero element in the $(k+1)$ -th column.

2.2. Linear inequalities

Throughout this subsection, \mathbb{F} denotes the field \mathbb{R} of real numbers, or the field \mathbb{Q} of rational numbers.

2.2.1. Basic theory

Given a system of linear inequalities $Ax \leq b, x \in \mathbb{F}^n$, the basic principle to derive a new inequality $c^T x \leq \delta$ is taking a *non-negative* linear combination of the given inequalities. Another possibility is *weakening* the right-hand side. We capture this by the following two inference rules:

$$\text{nonneg_lin_com} : \frac{Ax \leq b}{(u^T A)x \leq u^T b} \text{ if } \begin{cases} u \in \mathbb{F}^m, \\ u \geq 0 \end{cases} \quad (2.11)$$

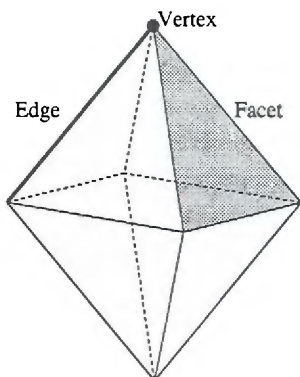


Figure 1: Polyhedron

$$\text{weak_rhs} : \frac{a^T x \leq \beta}{a^T x \leq \beta'} \text{ if } \beta \leq \beta' \quad (2.12)$$

Again there is a soundness and completeness result, which in the theory of linear optimization is known as *Farkas Lemma*. Any implied inequality of a solvable system can be obtained by one application of the rule `nonneg_lin_com` followed by one application of `weak_rhs`.

2.5. PROPOSITION (Solvability and entailment). *The system $Ax \leq b$ has no solution $x \in \mathbb{F}^n$ if and only if there exists $u \in \mathbb{F}^m, u \geq 0$ such that $u^T A = 0$ and $u^T b = -1$. If $Ax \leq b$ is solvable, then an inequality $c^T x \leq \delta$ is implied by $Ax \leq b$ in \mathbb{F}^n if and only if there exists $u \in \mathbb{F}^m, u \geq 0$ such that $u^T A = c^T$ and $u^T b \leq \delta$.*

The solution set of a linear inequality $a^T x \leq \beta, a \neq 0$ defines a *halfspace* in n -dimensional affine space \mathbb{F}^n . The solution set $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$ of a system of linear inequalities $Ax \leq b$, i.e. an intersection of finitely many halfspaces, is called a *polyhedron*. A bounded polyhedron is called a *polytope*. An inequality $c^T x \leq \delta$ is called *valid* for a polyhedron P if $P \subseteq \{x \in \mathbb{F}^n \mid c^T x \leq \delta\}$. A set $F \subseteq P$ is called a *face* of P if there exists an inequality $c^T x \leq \delta$ valid for P such that $F = P \cap \{x \in P \mid c^T x = \delta\}$. A point v in P such that $\{v\}$ is a face of P is called a *vertex* of P . A polyhedron is called *pointed* if it has a vertex. A non-empty polyhedron $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$ is pointed if and only if A has full column rank. A *facet* of P is an inclusionwise maximal face F with $\emptyset \neq F \neq P$. Equivalently, a facet is a nonempty face F of P with $\dim(F) = \dim(P) - 1$, see Fig. 1. If a polyhedron P is full-dimensional then P has a representation $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$ such that each inequality in $Ax \leq b$ defines a facet of P and such that each facet of P is defined by exactly one of these inequalities. If we normalize the inequalities, then this representation of P is unique.

In addition to the outer description of a polyhedron as the solution set of a linear inequality system, there is also an inner description, which is based on vertices and extreme rays. A vector $r \in \mathbb{F}^n$ is called a *ray* of the polyhedron P if for each $x \in P$ the set $\{x + \lambda r \mid \lambda \geq 0\}$ is contained in P . A ray r of P is *extreme* if there do not exist two linearly independent rays r^1, r^2 of P such that $r = \frac{1}{2}(r^1 + r^2)$.

2.6. THEOREM. *For each polyhedron $P \subseteq \mathbb{F}^n$ there exist finitely many points p^1, \dots, p^k in P and finitely many rays r^1, \dots, r^l of P such that*

$$P = \text{conv}(p^1, \dots, p^k) + \text{cone}(r^1, \dots, r^l). \quad (2.13)$$

If the polyhedron P is pointed then p^1, \dots, p^k may be chosen as the uniquely determined vertices of P , and r^1, \dots, r^l as representatives of the up to scalar multiplication uniquely determined extreme rays of P .

This result yields a parametric description of the solution set of a system of linear inequalities. In particular, every polytope can be described as the convex hull of its vertices.

Algebraically, the vertices of a polyhedron can be characterized as follows: Let $A \in \mathbb{F}^{m \times n}$ be a matrix of rank r . A non-singular $r \times r$ -submatrix B of A is called a *basis* of A . Let $A \in \mathbb{F}^{m \times n}$ have full column rank n and let $b \in \mathbb{F}^m$. For any basis $B = A_{I*}$ of A , let b_I be the subvector of b corresponding to B . Then the vector $B^{-1}b_I$ is called a *basic solution* of $Ax \leq b$. Note that $B^{-1}b_I$ need not satisfy $Ax \leq b$. If $B^{-1}b_I$ satisfies $Ax \leq b$, then it is called a *basic feasible solution* of $Ax \leq b$. A vector v is a vertex of the polyhedron $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$, A with full column rank, if and only if it is a basic feasible solution of $Ax \leq b$, for some basis B of A . Note that this basis need not be unique.

In many practical applications, one is interested in computing a solution of a system of linear inequalities that is optimal with respect to some linear objective function $f(x) = c^T x$, $c \in \mathbb{F}^n$. This leads to a *linear optimization* or *linear programming* problem. Given a matrix $A \in \mathbb{F}^{m \times n}$ and vectors $b \in \mathbb{F}^m$, $c \in \mathbb{F}^n$, find a vector $x^* \in P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$ maximizing the linear function $c^T x$ over P , i.e.

$$c^T x^* = \max\{c^T x \mid Ax \leq b, x \in \mathbb{F}^n\}. \quad (2.14)$$

A vector $x \in \mathbb{F}^n$ satisfying $Ax \leq b$ is called a *feasible* solution. A feasible solution x^* is called an *optimal solution* if $c^T x \leq c^T x^*$, for all feasible solutions $x \in P$. With every linear program of the form (2.14) we can associate another linear program

$$\min\{y^T b \mid y^T A = c^T, y \geq 0\}, \quad (2.15)$$

which is called the *dual* of the *primal* problem (2.14). Using trivial transformations, the dual program can be brought into the form (2.14).

2.7. THEOREM (Duality). *Let (P) $\max\{c^T x \mid Ax \leq b\}$ be a linear program and (D) $\min\{y^T b \mid y^T A = c^T, y \geq 0\}$ be its dual. If (P) and (D) both have feasible solutions,*

then both programs have optimal solutions and the optimum values of the objective functions are equal. If one of the programs (P) or (D) has no feasible solution, then the other is either unbounded or has no feasible solution. If one of the programs (P) or (D) is unbounded, then the other has no feasible solution.

There exist various different formulations of linear optimization problems, which are all equivalent. One possibility is the *standard form*

$$c^T x^* = \max\{c^T x \mid Ax = b, x \geq 0\}. \quad (2.16)$$

To obtain this form from (2.14), we can replace each inequality $a^T x \leq \beta$ by the equation $a^T x + x' = \beta$ using a new variable $x' \geq 0$, which is called a *slack variable*. Furthermore, any unbounded variable x in (2.14) can be written as the difference $x = x^+ - x^-$ of two new non-negative variables $x^+, x^- \geq 0$.

2.2.2. The Simplex method

The classical method for solving linear programs is the Simplex algorithm, which was proposed by Dantzig in 1947. While this method is usually described for linear programs in standard form, we follow here the presentation in [Grötschel et al. 1988] and consider a linear program of the form

$$\max\{c^T x \mid Ax \leq b, x \in \mathbb{F}^n\}. \quad (2.17)$$

Assume first that the matrix $A \in \mathbb{F}^{m \times n}$ has full column rank. This implies that the polyhedron $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$ has a vertex. In geometric terms, the Simplex method can be described as follows. We start at a vertex of P and check whether there is an edge leaving this vertex along which the objective function can be increased. If there is no such edge, the current vertex is optimal. Otherwise, we follow one of these edges. Either we find a new vertex, in which case we can repeat the procedure, or the edge is infinite, in which case the linear program (2.17) is unbounded.

We now translate these geometric ideas into an algebraic framework. A *basic feasible index set* is a row index set $I \subseteq \{1, \dots, m\}$ of cardinality n for which A_{I*} is a basis of A and for which $A_{I*}^{-1}b_I$ is feasible. As we have mentioned before, each vertex v of P can be represented by a basic feasible index set I such that $v = A_{I*}^{-1}b_I$. Instead of creating a sequence of vertices, the Simplex method constructs a sequence I_1, I_2, \dots of basic feasible index sets. Remember, however, that this representation of a vertex need not be unique. Therefore, in the sequence $A_{I_1*}^{-1}b_{I_1}, A_{I_2*}^{-1}b_{I_2}, \dots$ two consecutive vertices need not be different.

We now describe one iteration of the Simplex method. Suppose we have a basic feasible index set I with corresponding vertex $v = A_{I*}^{-1}b_I$. We calculate

$$u^T := c^T A_{I*}^{-1}. \quad (2.18)$$

Note that u is an n -dimensional vector indexed by I . We distinguish two cases:

1. If $u \geq 0$, then v is an optimal solution of (2.17), because for each feasible solution x

$$c^T x = u^T A_{I*} x \leq u^T b_I = u^T A_{I*} v = c^T v. \quad (2.19)$$

2. If $u \not\geq 0$, we choose $i \in I$ such that $u_i < 0$ and define the direction

$$d := -A_{I*}^{-1} e_i, \quad (2.20)$$

where e_i is the i -th unit basis vector in \mathbb{F}^I .

The idea is to increase the value of the objective function by going from v in direction d while maintaining feasibility.

Note that $c^T(v + \lambda d) = c^T v - \lambda u^T e_i = c^T v - \lambda u_i > c^T v$, for $\lambda > 0$.

- (a) If $Ad \not\leq 0$, then the largest $\lambda \geq 0$ for which $v + \lambda d$ is still feasible is given by

$$\lambda^* = \min \left\{ \frac{b_l - A_{l*} v}{A_{l*} d} \mid l \in \{1, \dots, m\}, A_{l*} d > 0 \right\}. \quad (2.21)$$

Let this minimum be attained at index k . Then $k \notin I$ because $A_{I*} d = -e_i \leq 0$. We define the new basic feasible index $I' = (I \setminus \{i\}) \cup \{k\}$, which corresponds to the vertex $v + \lambda^* d$. Then we replace I by I' and repeat the iteration.

- (b) If $Ad \leq 0$, then $v + \lambda d$ is feasible, for all $\lambda \geq 0$. Moreover,

$$c^T d = -c^T A_{I*}^{-1} e_i = -u^T e_i = -u_i > 0. \quad (2.22)$$

Thus the objective function can be increased along d to infinity and (2.17) is unbounded.

The method terminates if the indices i and k are chosen in the right way (such choices are called *pivoting rules*). For example, following the rule of Bland [1977], we can always choose the smallest i such that $u_i < 0$ and the smallest k attaining the minimum in (2.21). For most known pivoting rules, sequences of examples have been constructed such that the number of iterations is exponential in $m + n$ [Klee and Minty 1972]. Although no pivoting rule is known to yield a polynomial time algorithm, the Simplex method turns out to work very well in practice. Borgwardt [1982] proved that in some natural probabilistic model, and with an appropriate pivoting rule, the expected running time of the Simplex method is polynomial.

In the general case, where we know nothing about the matrix A , we may first apply Gaussian elimination in order to satisfy the hypothesis that A has full column rank. In order to find an initial basic feasible index set, we can consider an auxiliary linear program

$$\max \{y \mid \begin{array}{rcl} Ax - by & \leq & 0 \\ -y & \leq & 0 \\ y & \leq & 1 \end{array} \}, \quad (2.23)$$

where y is a new variable. Given an arbitrary basis A_{K*} of A , we obtain a basic feasible index set I for (2.23) by choosing $I = K \cup \{m+1\}$. The corresponding basic feasible solution is 0. Now, we apply the Simplex method to (2.23). If the optimum value is 0, then (2.17) is infeasible. Otherwise, the optimum value has to be 1. If I' is the final basic feasible index set of (2.23), then $K' = I' \setminus \{m+2\}$ can be used as an initial basic feasible index set for (2.17).

2.2.3. The ellipsoid method and the complexity of linear programming

For a long time, it was an open question whether there is a polynomial algorithm for solving linear inequalities over the rational numbers. Only in 1979, Khachiyan showed that the *ellipsoid method* for nonlinear programming can be adapted to solve linear inequalities and linear programming in polynomial time.

2.8. THEOREM (Khachiyan 79). *The following problems are solvable in polynomial time:*

- Given a matrix $A \in \mathbb{Q}^{m \times n}$ and a vector $b \in \mathbb{Q}^m$, decide whether $Ax \leq b$ has a solution $x \in \mathbb{Q}^n$, and if so, find one.
- Given a rational matrix $A \in \mathbb{Q}^{m \times n}$ and a rational vector $b \in \mathbb{Q}^m$, decide whether $Ax = b$ has a nonnegative solution $x \in \mathbb{Q}^n, x \geq 0$, and if so, find one.
- (Linear programming problem) Given a matrix $A \in \mathbb{Q}^{m \times n}$ and vectors $b \in \mathbb{Q}^m, c \in \mathbb{Q}^n$, decide whether $\max\{c^T x \mid Ax \leq b, x \in \mathbb{Q}^n\}$ is infeasible, finite, or unbounded. If it is finite, find an optimal solution. If it is unbounded, find a feasible solution x_0 , and find a vector $z \in \mathbb{Q}^n$ with $Az \leq 0$ and $c^T z > 0$.

A set $E \subseteq \mathbb{R}^n$ is called an *ellipsoid* if there exists a vector $c \in \mathbb{R}^n$, which is called the *center* of E , and a positive definite matrix $D \in \mathbb{R}^{n \times n}$ such that

$$E = E(c, D) = \{x \in \mathbb{R}^n \mid (x - c)^T D^{-1} (x - c) \leq 1\}. \quad (2.24)$$

Remember that a symmetric matrix $D \in \mathbb{R}^{n \times n}$ is *positive definite*, if $x^T D x > 0$, for all $x \in \mathbb{R}^n, x \neq 0$. Any positive definite matrix is non-singular, and D^{-1} is again positive definite. The unit ball $B(0, 1) = \{x \in \mathbb{R}^n \mid x_1^2 + \dots + x_n^2 \leq 1\}$ around 0 in the Euclidean norm is identical with the ellipsoid $E(0, I)$. For every positive definite matrix D there exists a unique positive definite matrix $D^{1/2}$ such that $D = D^{1/2} D^{1/2}$. It follows that $E(c, D) = D^{1/2} B(0, 1) + c$. Thus every ellipsoid is the image of the unit ball under a bijective affine transformation. The *volume* $\text{vol}(E)$ of an ellipsoid $E = E(c, D)$ is given by $\text{vol}(E(c, D)) = \sqrt{\det D} \cdot V_n$, where V_n denotes the volume of the unit ball $B(0, 1)$ in \mathbb{R}^n .

We now give an outline of the ellipsoid method for deciding whether a system of linear inequalities $Ax \leq b$ with integer coefficients is solvable over the real or rational numbers. Consider the polyhedron

$$P = \{x \in \mathbb{F}^n \mid Ax \leq b\}, \quad A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m, \quad (2.25)$$

and let us assume that P is either empty or bounded and full-dimensional. The basic idea is to construct a sequence of ellipsoids $E_t, t \in \mathbb{N}$, such that all E_t contain P and such that $\text{vol}(E_t)$ gets smaller and smaller. Suppose that, in the t -th iteration, we have computed the ellipsoid $E_t = E(c_t, D_t)$. If $c_t \in P$, then P is non-empty and the algorithm terminates. If $c_t \notin P$, there exists an inequality $a^T x \leq \beta$ in the system $Ax \leq b$ such that $a^T c_t > \beta$. It follows that P is contained in the intersection $E_t \cap H$ of E_t with the half-space $H = \{x \in \mathbb{R}^n \mid a^T x \leq a^T c_t\}$. Now we can construct a new ellipsoid E_{t+1} containing the half-ellipsoid $H \cap E_t$ such that the volume of E_{t+1} is only a fraction of the volume of E_t , see Fig. 2 for illustration.

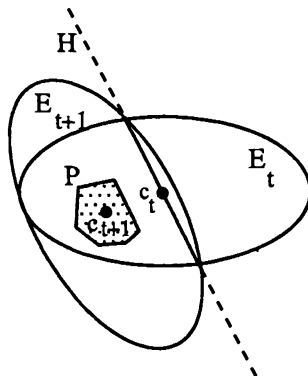


Figure 2: One iteration of the ellipsoid method

2.9. THEOREM. Let $E_t = E(c_t, D_t) \subseteq \mathbb{R}^n$ be an ellipsoid and let $a \in \mathbb{R}^n$ be a non-zero vector. Consider the halfspace $H = \{x \in \mathbb{R}^n \mid a^T x \leq a^T c_t\}$ defined by the hyperplane in direction a through c_t . Let

$$c_{t+1} = c_t - \frac{1}{n+1} d_t \quad \text{and} \quad D_{t+1} = \frac{n^2}{n^2 - 1} \left(D_t - \frac{2}{n+1} d_t d_t^T \right), \quad (2.26)$$

where

$$d_t = \frac{1}{\sqrt{a^T D_t a}} D_t a. \quad (2.27)$$

Then $E_{t+1} = E(c_{t+1}, D_{t+1})$ is an ellipsoid such that

- $E_t \cap H \subset E_{t+1}$
- $\text{vol}(E_{t+1}) < e^{-\frac{1}{2n}} \text{vol}(E_t)$

Repeating this process, we either find a point in P or we conclude that the volume of P is smaller than some lower bound on the volume of P , if P is nonempty. This implies that P is empty. One can show that if P is full-dimensional, then $\text{vol}(P) \geq 2^{-(n+1)\langle A \rangle + n^3}$, where $\langle A \rangle$ is the encoding length of A . An initial ellipsoid $E_0 = E(0, R^2 I)$ that contains a point of P , if P is non-empty, can be obtained by observing that all vertices of P , if any, are contained in the ball around 0 with radius $R = \sqrt{n} 2^{\langle A, b \rangle - n^2}$. Remember that a non-empty P has a vertex if and only if A has full column rank. Combining the previous results one can see that after

$$N = 2n((2n+1)\langle A \rangle + n\langle b \rangle - n^3) \quad (2.28)$$

iterations of the ellipsoid method, either we have found a point in P or we can conclude that P is empty because $\text{vol}(E_N) < \text{vol}(P)$. In order to prove that the ellipsoid method with rational arithmetic runs in polynomial time, we must also show that the square root in (2.27) can be approximated in polynomial time without

losing correctness of the algorithm, and that the intermediate rational numbers do not get too large. These results are rather technical and do not offer much insight, see [Schrijver 1986, Grötschel et al. 1988] for more details. The assumption that P is full-dimensional, if it is not empty, can be removed by replacing P with the polyhedron $P_\epsilon = \{x \in \mathbb{R}^n \mid Ax \leq b + \epsilon \cdot e\}$, where $\epsilon = \frac{1}{2n} 2^{-4n^2 \langle A, b \rangle}$ and $e = (1, \dots, 1)^T \in \mathbb{R}^m$. Then P_ϵ is empty if P is empty, and P_ϵ is full-dimensional, if P is non-empty.

The ellipsoid method is an important theoretical tool for analyzing the complexity of solving linear arithmetic constraints. Unfortunately, it does not result in an algorithm that is useful in practice. The convergence rate is slow and the precision demands are very high. Therefore, from a practical perspective, the theoretically inefficient Simplex algorithm behaves much better.

The ellipsoid method is not strongly polynomial. The number of elementary arithmetic operations depends on the encoding length of the input data. Tardos [1986] showed that it is possible to solve any linear program $\max\{c^T x \mid Ax \leq b\}$ in $p(\langle A \rangle)$ elementary arithmetic operations on numbers with encoding length polynomially bounded by $\langle A, b, c \rangle$, where p is a polynomial. In other words, the number of elementary arithmetic operations does not depend on $\langle b \rangle$ and $\langle c \rangle$. This implies that combinatorial linear programs, where A is a $\{0, \pm 1\}$ matrix, have a strongly polynomial algorithm. In [Frank and Tardos 1987], this result is extended to problems, where the number of inequalities may be exponential in the dimension. Megiddo [1984] proved that for each fixed n there is an algorithm for solving linear programs in n variables and m constraints that requires only $O(m)$ elementary arithmetic operations on numbers with polynomial encoding length. Combinatorial algorithms for linear programming in two variables per inequality have been developed among others in [Cohen and Megiddo 1994], [Hochbaum and Naor 1994], and [Wayne 1999].

2.2.4. Interior point methods

The Simplex method solves a linear optimization problem by visiting extreme points, all lying on the boundary of the corresponding polyhedron. *Interior point algorithms* find an optimal solution while moving through the relative interior of the feasible region. They were first described by Dikin [1967], current interest started with a new polynomial-time algorithm for linear programming by Karmarkar [1984]. Interior point methods combine the advantages of the Simplex algorithm and the ellipsoid method. From a theoretical point of view, many interior point algorithms have polynomial complexity like the ellipsoid method. In practice, however, they behave much better. They are competitive with the Simplex method and sometimes even superior, in particular when solving linear optimization problems with a large number of variables and constraints. There exists a variety of interior point methods based on different geometric ideas. We can sketch here only some of the key concepts and refer to the literature for a more detailed description [Bertsimas and Tsitsiklis 1997, Vanderbei 1997, Ye 1997].

- *The affine scaling algorithm* is probably the simplest interior point algorithm. However, it has already good practical performance. The basic idea is to con-

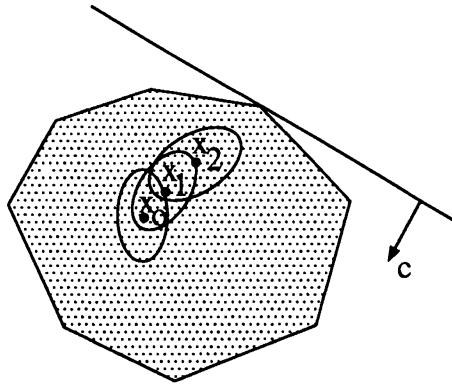


Figure 3: The affine scaling algorithm

struct a sequence of ellipsoids lying inside the feasible region such that the centers of these ellipsoids converge to an optimal solution (Fig. 3).

- The *potential reduction algorithm* measures the progress towards optimality by the reduction in a certain nonlinear potential function, which attempts to balance two conflicting objectives: improving the objective function value and staying away from the boundary of the feasible set.
- *Path following algorithms* first transform the constrained linear program to an unconstrained one, by incorporating the constraints in a *logarithmic barrier function* that imposes a growing penalty as one gets closer to the boundary. Then the unconstrained problem is solved approximately by applying Newton's method. As the strength of the barrier function is decreased, the optimum of the unconstrained problem follows a path that ends at an optimal solution of the original problem.

We describe briefly the *primal-dual path following algorithm*, which may be the most popular interior-point method. It has excellent performance in large scale applications and is used in various commercial implementations. We start with a linear programming problem in standard form

$$\min\{c^T x \mid Ax = b, x \geq 0\} \quad (2.29)$$

and its dual

$$\max\{y^T b \mid y^T A + z^T = c^T, z \geq 0\}. \quad (2.30)$$

For any positive constant $\mu > 0$, we consider the *barrier optimization problem*

$$\min\{c^T x - \mu \sum_{j=1}^n \log(x_j) \mid Ax = b\}. \quad (2.31)$$

Here, \log denotes the natural logarithm. Note that if some x_j approaches 0, then the objective function value goes to $+\infty$. The objective function is defined to be $+\infty$,

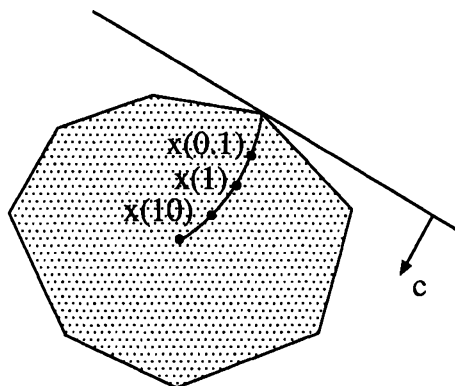


Figure 4: The central path

if some $x_j \leq 0$. The barrier problem has a unique solution $x(\mu)$ if the original linear program has a compact set of optimal solutions. The set of all optimal solutions for varying μ is known as the *central path*. It can be shown that $x^* = \lim_{\mu \rightarrow 0} x(\mu)$ exists and is an optimal solution for the original linear program, see Fig. 4.

The barrier problem from the dual problem is

$$\max\{y^T b + \mu \sum_{j=1}^n \log(z_j) \mid y^T A + z^T = c^T\}. \quad (2.32)$$

The optimal solutions $(x(\mu), y(\mu), z(\mu))$ to the barrier problems (2.31) and (2.32) have to satisfy the *Karush-Kuhn-Tucker conditions*

$$\begin{aligned} Ax(\mu) &= b \\ x(\mu) &\geq 0 \\ A^T y(\mu) + z(\mu) &= c \\ z(\mu) &\geq 0 \\ X(\mu)Z(\mu)e &= \mu e, \end{aligned} \quad (2.33)$$

where $X(\mu) = \text{diag}(x_1(\mu), \dots, x_n(\mu))$, $Z(\mu) = \text{diag}(z_1(\mu), \dots, z_n(\mu))$ are diagonal matrices and $e = (1, \dots, 1)^T$.

The path-following method starts with a feasible solution (x, y, z) close to the point $(x(\mu), y(\mu), z(\mu))$ on the central path. The basic idea is to decrease the value of μ to $\bar{\mu}$ and to apply Newton's method in order to obtain a new feasible solution $(\bar{x}, \bar{y}, \bar{z})$ close to $(x(\bar{\mu}), y(\bar{\mu}), z(\bar{\mu}))$, see Fig. 5. The iterative update of the solution (x, y, z) involves the following steps:

1. Determine an appropriate value for $\bar{\mu}$.
2. Apply Newton's method to the non-linear system (2.33) in order to obtain step directions $(\Delta x, \Delta y, \Delta z)$ pointing approximately to the point $(x(\bar{\mu}), y(\bar{\mu}), z(\bar{\mu}))$ on the central path.

Using the trick of the “sliding objective function”, the method can be used to solve linear programming (LP) problems $\min\{c^T x \mid Ax \leq b\}$. Form a new system $Ax \leq b, c^T x \leq z$ of linear constraints with an additive parameter z . Then the elimination of all variables x will result in a system of linear constraints in the single parameter z that can be solved easily. The minimal feasible value of z will then agree with the minimal value z_0 of $c^T x$ under the given constraints. Plugging in this value z_0 for z in the next to last system, we can determine a corresponding value of x_n . Plugging in this value for x_n in the previous system yields a value for x_{n-1} and so on. In this manner we get the coordinates of a minimum point for $c^T x$ under the given constraints.

The major drawback of the method for practical purposes is the rapid increase of the number of constraints that happens during elimination: If in the notation above we have $k = m/2$, then the elimination of the variable x_1 from an input system consisting of m inequalities results in a system of $m^2/4$ inequalities. In the worst-case, this phenomenon can happen roughly in every elimination step, resulting in a doubly exponential increase of the number of inequalities. As a result the worst-case complexity of this method is doubly exponential in the number of variables, even if the computations on the coefficients are taken at unit cost [Weispfenning 1994].

So it is not surprising that - despite some optimization strategies due e.g. to Černikov, Kohler and Duffin [Černikov 1963, Williams 1986, Duffin 1974] - the method is usually unable to solve linear optimization problems of interesting size. Only for problems of very special type, e.g. at most two variables in each inequality there are polynomial upper bounds, see [Chandru 1993] and the references given there. From the theoretical viewpoint, however, the method remains important, since it provides a simple approach to basic structural theorems of LP such as the feasibility theorem, the Farkas Lemma and the Duality Theorem [Dantzig and Eaves 1973, Williams 1986]. An implementation is available e.g. in the linear algebra package of REDUCE. Other useful side-products of the Fourier-Motzkin method are the discovery of equations that are implicitly implied by a system of linear inequalities, and of inequalities that make the system inconsistent [Huynh, Joskowicz, Lassez and Lassez 1990, Lassez 1990].

As a more efficient alternative to Fourier-Motzkin elimination [Huynh et al. 1990, Lassez 1990] propose variable elimination in systems of linear inequalities via extreme point computation. This method eliminates a whole set of variables at once. By forming a quasi-dual of the given system S of linear constraints with respect to the variables to be eliminated, one arrives at a generalized linear optimization problem with a vector valued objective function Φ ranging over a bounded polyhedron P . Then the extreme points of the polytope $\Phi(P)$ determine a finite system of linear inequalities that describe the projection of the solution set of S into the space of the remaining variables.

2.2.6. Elimination by substitution of test points

In this section we describe another elimination method for systems of linear constraints that is both asymptotically and practically much more efficient than the

Fourier-Motzkin method. It was developed in [Weispfenning 1988, Weispfenning 1994, Loos and Weispfenning 1993, Weispfenning 1997c] and implemented in [Burhenne 1990, Kappert 1994], and in the REDLOG-package of REDUCE [Dolzmann and Sturm 1996, Dolzmann and Sturm 1997].

As in the Fourier-Motzkin approach described in the previous section, a given system S of linear constraints of the form $Ax \leq b$ is prepared for the elimination of x_1 by transforming it into an equivalent system S' together with constraints of the form $f_i \leq x_1$ ($1 \leq i \leq k$), $x_1 \leq g_j$ ($k < j \leq m$), where S' is a system of linear constraints for the variables (x_2, \dots, x_n) , f_i, g_j are linear polynomials in the remaining variables (x_2, \dots, x_n) . For fixed real values of the variables (x_2, \dots, x_n) , the latter system determines a closed interval for the possible real values of x_1 . This interval is non-empty iff one of the possible lower endpoints f_i is contained in it, or equivalently iff one of the possible upper endpoints g_j is contained in it. If no lower bound f_i (or no upper bound g_j) is present, then the interval is always non-empty. So if the given system S has a real solution (x_1, \dots, x_n) , then for some $1 \leq i \leq k$, $(f_i(x_2, \dots, x_n), x_2, \dots, x_n)$ will also be a solution; similarly for some $k < j \leq m$, $(g_j(x_2, \dots, x_n), x_2, \dots, x_n)$ will also be a solution. In other words, the given system S is feasible iff for some $1 \leq i \leq k$ the system S_i obtained from S by substituting $f_i(x_2, \dots, x_n)$ for x_1 is feasible; similarly for the systems obtained from S by substituting some $g_j(x_2, \dots, x_n)$ for x_1 . If no f_i (g_j) is present one may simply substitute $-\infty$ (∞) for x_1 . Notice that the variable x_1 has been eliminated from each of the new systems S_i . So the same elimination step can now be applied to the variable x_2 in each S_i yielding a new finite set of systems $S_{i,j}$ of linear constraints in x_3, \dots, x_n . Iterating this procedure, we arrive at a finite tree of such systems, whose bottom nodes are finite systems of inequalities between real constants, and hence can be evaluated to "true" or "false". The original system S has a real solution iff at least one of the bottom nodes is evaluated to "true".

In case of a linear optimization or feasibility problem, the elimination of a single variable by this method corresponds geometrically to a passage from the polyhedron P described by the linear constraints to the projections along a coordinate axis of some of its facets or the empty set. During the elimination of further variables the different facets produced earlier do not interact, but are treated separately. It is only after the completion of the elimination process that the different values of the objective function at the vertices of P produced by the elimination process are compared. This should be contrasted to the Fourier-Motzkin method, where elimination of a variable corresponds to the projection of all of P along a coordinate axis. So, roughly speaking, the method of substitution of test points is more "local" in character than the "global" Fourier-Motzkin method. This feature offers the additional advantage that the method is well parallelizable. First steps towards parallelization have been quite encouraging [Dolzmann, Gloor and Sturm 1998].

Another feature of the method is the fact that it can handle *arbitrary Boolean combinations of weak or strict linear inequalities* which may in addition include additive parameters: For "and"- "or"-combinations $\varphi(x)$ of weak linear inequalities one proceeds exactly as described above for conjunctive systems. It is not necessary to expand the given "and"- "or"-combination $\varphi(x)$ into disjunctive normal form;

this avoids the possibly exponential size explosion involved in this step. The method yields a finite number of test points t_i , $i \in I$ such that $\varphi(x)$ has a solution iff some t_i satisfies $\varphi(x)$. If $\varphi(x)$ contains additive parameters, the test points t_i , $i \in I$ will depend on these parameters. The characteristic property of the test points will then hold for every real value of the parameters.

If one admits in such an “and”-“or”-combination of constraints also strict linear inequalities, one requires additional test points of the form $f_i \pm \epsilon$, where the f_i are the test points used earlier and ϵ is a positive “*infinitesimal*”. The substitution of a test point of this form into the given constraints can be performed in such a way that the result does no longer contain the symbol ϵ . The same applies to the substitution of test points of the form $\pm\infty$. Hence the method is also referred to as *virtual substitution of test points* (compare Section 2.4.2).

Another extension of the method concerns Boolean combinations of quadratic inequalities (see Section 4.5).

2.3. Disequalities

A *positive linear constraint* is a conjunction

$$Ex = f, Ax \leq b \quad (2.34)$$

of linear equations and inequalities. A *negative linear constraint* is a disjunction of disequalities $c_i^T x \neq \delta_i$, $i = 1, \dots, m$, which will be written as $\neg Cx = d$, with a matrix $C = (c_1, \dots, c_m)^T \in \mathbb{F}^{m \times n}$ and a vector $d = (\delta_1, \dots, \delta_m) \in \mathbb{F}^m$. A system of *generalized linear constraints* [Lassez and McAloon 1992] is a finite conjunction of positive and negative linear constraints. We will use the notation

$$Ex = f, Ax \leq b, \{\neg C_k x = d_k\}_{k=1}^l. \quad (2.35)$$

The key for handling generalized linear constraints is the *independence of negative constraints*. The system (2.35) is feasible if and only if the positive constraints (2.34) together with one negative constraint $\neg C_k x = d_k$ are feasible, for all $k = 1, \dots, l$. Geometrically, this means that a polyhedron is contained in a finite union of affine sets if and only if it is contained in one of them. Suppose the positive constraints (2.34) are feasible and let $\text{Aff}(P)$ denote the affine closure of their solution set P . Lassez and Maher [1992] show that Fourier elimination can be easily modified to compute the affine hull of a polyhedron. The negative constraint $\neg Cx = d$ is consistent with (2.34) if and only if $\text{Aff}(P) \not\subseteq \{x \mid Cx = d\}$. Lassez and McAloon [1992] defined the following notion of *canonical form* for a system of generalized linear constraints:

- the linear equations $Ex = f$ are in *solved form*, i.e. x can be partitioned into (x_E, x_P) such that $Ex = f$ has the form $x_E = \tilde{E}_P x_P + \tilde{f}$
- the linear inequalities $Ax \leq b$ define a full-dimensional polyhedron P in x_P -space. Here, all implicit equalities and redundant inequalities have to be removed such that each inequality in $Ax \leq b$ defines exactly one facet of P .

- the negative constraints $\neg Cx = d$ are *precise*, i.e. for $A = \{x \mid Cx = d\}$ we have $A \cap P \neq \emptyset, A = \text{Aff}(A \cap P)$, and the complementary linear equations $Cx = d$ are again in solved form.

They showed that, using linear optimization, the canonical form can be computed in polynomial time. This implies that one can decide in polynomial time whether a system of generalized linear constraints (with rational coefficients over the rational numbers) is feasible, and, if so, compute a rational solution. Koubarakis [1996] extended this result to disjunctions with a finite number of disequalities and at most one inequality.

A lot of work on linear arithmetic constraint solving has been done in the area of constraint logic programming [Colmerauer 1987, Jaffar and Lassez 1987, Dincbas, van Hentenryck, Simonis, Aggoun and Graf 1988, Imbert 1995b]. While in $\text{CLP}(\mathcal{R})$, linear constraints are solved over floating point numbers, the constraint solvers of PROLOG III/IV and CHIP provide infinite precision rational arithmetic, at the expense of efficiency. Holzbaur [1995] has developed a PROLOG-based solver for real and rational linear arithmetic, which has been integrated in the constraint logic programming systems SICStus and Eclipse.

Linear arithmetic constraint solvers in constraint logic programming are based on the Simplex algorithm, which has to be modified in order to accommodate the requirements of incrementality and non-determinism, and the presence of disequalities [Stuckey 1991, Huynh and Marriott 1995]. An in depth description of the $\text{CLP}(\mathcal{R})$ language and system can be found in [Jaffar, Michaylov, Stuckey and Yap 1992], the solved form used in the CHIP system is described in [van Hentenryck and Graf 1992]. Variable elimination in $\text{CLP}(\mathcal{R})$ is discussed in [Huynh, Lassez and Lassez 1992, Jaffar, Maher, Stuckey and Yap 1993] and, with special emphasis, on disequalities in [Imbert 1993, Imbert 1997]. An empirical evaluation of different constraint solvers for $\text{CLP}(\mathcal{R})$ is given in [Burg, Stuckey, Tai and Yap 1995], while Imbert [1995a] compares and unifies several variable elimination methods for systems of linear inequalities that are based on Fourier's approach. Azulay and Pique [1998] study the use of Q-pivots for exact linear solvers.

2.4. Extensions

2.4.1. Parametric linear programming

A linear programming problem given by a system $Ax \leq b$ of linear constraints and an objective function $f(x) = c^T x$ is *parametric* if the data, i.e. the matrix A and the vectors b, c involve real parameters that may take on arbitrary real values. Typical instances are data that depend on a time parameter, or problems where some of the data may vary frequently, and hence should not be fixed in advance. The goal is either to get optimal solutions for all parameter values in some given range, or to perform a *sensitivity analysis*. In the latter case, one is interested in parameter values near a given point p in the parameter space; the goal is then to describe a region of parameter values containing p , where the optimal solution depends smoothly on the parameter values in some explicit way.

The elimination methods described in Section 2.2.5 and 2.2.6 are well-suited for parametric problems, since they are inherently parametric. Call the given linear programming problem *weakly parametric* if parameters occur only in the vector b and *strongly parametric* otherwise. In other words weakly parametric problems involve only additive parameters, while strongly parametric problems contain multiplicative parameters.

Both Fourier-Motzkin elimination and elimination by test points can readily cope with weakly parametric problems. The output of Fourier-Motzkin elimination is a conjunction ψ of linear inequalities involving the parameters and a new variable z representing the “sliding objective function”. For given parameter values the minimum of the objective function under the given constraints equals the minimal real value of z satisfying ψ . The output of elimination by test points is a list of triples consisting of a conjunction ψ of linear inequalities involving the parameters and z , a linear function q of the parameters and of z , and a tuple p of linear functions of the parameters and of z . For given parameter values the minimum value of z can be determined in each first entry of the triple, yielding a corresponding value of the objective function and point coordinates. By comparing the values of the objective function in each triple one finds a minimum value and the coordinates of a minimum point. This output can also easily be adapted to a sensitivity analysis near given parameter values.

As pointed out in Section 2.2.5, Fourier-Motzkin elimination is as a rule too complex to deal with weakly parametric problems in practice, whereas elimination by test points can handle problems of significant size. Both methods can also be adapted to the strongly parametric case by case distinctions on the parameters [Eaves and Rothblum 1992, Weispfenning 1994].

2.4.2. Linear quantifier elimination

The linear theory T_{lin} of real numbers is the first-order theory of the reals as an ordered linear space over the rational numbers. So the language L_{lin} of T has 0 as a constant, scalar multiplication with rational numbers as unary operations, $+$, $-$ as binary operations, and equality $=$ and order \leq as binary relations. L_{lin} -terms can be written in the form of linear polynomials with rational coefficients. Atomic formulas are equations and inequalities between such L -terms. T_{lin} can be construed as the “linear fragment” of the first-order theory of the reals considered in Section 4.4. This relatively small fragment is nevertheless rich enough to express many important application problems, such as feasibility of linear constraints, linear programming with additive parameters, scheduling problems, simulation of electrical networks [Weispfenning 1997c, Sturm 1997].

Explicit algorithmic quantifier elimination procedures for T_{lin} are obtained both from slight extensions of Fourier-Motzkin elimination Section 2.2.5 and elimination by virtual substitution of test points Section 2.2.6.

2.10. THEOREM. T_{lin} admits quantifier elimination in L_{lin} in time $\ell^{(cn)^b}$ for some constant c . The validity of a closed formula can be decided in time $\ell^{(cn)^b}$ for some

constant c . Here ℓ is the length of the prenex input formula φ , n the number of quantified variables in φ and b the number of quantifier-blocks in φ .

The complexity of the decision problem is a direct consequence of the complexity of the quantifier elimination problem [Weispfenning 1988]. By a modified quantifier elimination procedure a sharper result was obtained in [Sontag 1985] for formulas with a bounded number b of quantifier blocks: The decision problem in T_{lin} for prenex formulas with b quantifier blocks beginning with an existential block is in the class Σ_b^P of the polynomial hierarchy. So the decision problem in T_{lin} for existential formulas is in NP; this special case was observed already in [Gathen and Sieveking 1976].

In order to prove quantifier elimination it suffices to eliminate one quantifier at a time. Moreover, since universal quantification can be expressed by existential quantification and negation via the equivalence

$$\forall x(\varphi) \iff \neg \exists x \neg(\varphi), \quad (2.36)$$

it suffices to eliminate one existential quantifier in front of a quantifier-free formula φ . By putting φ into disjunctive normal form and interchanging the existential quantifier with the disjunction we are reduced to the special case of one existential quantifier in front of a conjunction of equations and inequalities. Separating the quantified variable x on the left hand side of the equations and inequalities we arrive at an input formula of the form

$$\exists x \left(\bigwedge_{i=1}^m x \rho_i t_i \right), \quad (2.37)$$

where t_i are terms not containing the variable x , and ρ_i is one of the relations "=", "<", ">". If this formula contains at least one equation, say $x = t_1$, then it is equivalent to

$$\bigwedge_{i=2}^m (t_1 \rho_i t_i). \quad (2.38)$$

Otherwise it is essentially of the form

$$\exists x \left(\bigwedge_{i=1}^k x > t_i \wedge \bigwedge_{j=k+1}^m x < t_j \right). \quad (2.39)$$

If $k = 0$ or $m = k$ then this formula is obviously true. Otherwise, quantifier elimination via the Fourier-Motzkin method produces the equivalent formula

$$\bigwedge_{i=1}^k \bigwedge_{j=k+1}^m t_i < t_j. \quad (2.40)$$

In contrast, elimination via substitution of test points produces the equivalent formula

$$\bigvee_{r=1}^k \bigvee_{s=k+1}^m \bigwedge_{i=1}^k \frac{1}{2}(t_r + t_s) > t_i \wedge \bigwedge_{j=k+1}^m \frac{1}{2}(t_r + t_s) < t_j. \quad (2.41)$$

The advantage of the second method is the fact that subsequent further existential quantifiers can be interchanged with the disjunction produced in this elimination, whereas the first method yields one large conjunction. Under iteration the second method thus produces the indicated complexity behaviour, whereas the first is much more complex [Weispfenning 1988, Weispfenning 1994, Weispfenning 1997c]. In fact the second method can be improved considerably in practical complexity, when one admits virtual substitution of test points (compare Section 2.2.6). Then the output will be the shorter of the following two expressions:

$$\bigvee_{r=1}^k \bigwedge_{i=1}^k t_r + \epsilon > t_i \wedge \bigwedge_{j=k+1}^m t_r + \epsilon < t_j. \quad (2.42)$$

$$\bigvee_{s=k+1}^m \bigwedge_{i=1}^k t_s - \epsilon > t_i \wedge \bigwedge_{j=k+1}^m t_s - \epsilon < t_j. \quad (2.43)$$

Here ϵ denotes a positive infinitesimal number that can be eliminated easily from these expressions yielding again quantifier-free formulas [Loos and Weispfenning 1993].

A side-effect of elimination by test points is the fact that this method yields sample answers for existentially quantified variables. This feature is of importance in many applications, especially in optimization problems [Weispfenning 1994, Weispfenning 1997c].

As a by-product an analysis of the properties required in the course of these quantifier elimination procedures shows that only the axioms of ordered linear spaces over the field of rational numbers are required. So quantifier elimination and the induced decision procedure are valid in all such spaces in particular in all subfields of the reals:

2.11. COROLLARY. *T_{lin} is the common first-order theory of all ordered linear spaces over the field of rational numbers in L_{lin} .*

The upper bounds for the quantifier elimination and decision problem in T_{lin} have closely matching worst-case lower bounds. The quantifier elimination problem for T_{lin} is inherently doubly exponential in the worst-case [Weispfenning 1988]. More precisely: There is a series $\{\varphi_n\}$ of L -formulas of length linear in n such that for any T_{lin} -equivalent series of quantifier-free formulas $\{\psi_n\}$ the length of ψ_n grows doubly exponentially with n . For formulas with a bounded number b of quantifier blocks there is a corresponding refined singly exponential lower bound, where the second exponent equals b . The decision problem in T_{lin} for prenex closed formulas with a bounded number b of quantifier blocks starting with a block of existential quantifiers is complete for the class Σ_b^P of the polynomial hierarchy; in particular the decision problem in T_{lin} for closed existential formulas is NP-complete [Sontag 1985, Fürer 1982]. As a consequence the decision problem in T_{lin} for closed universal formulas is coNP-complete. Using conjunctive normal forms any closed universal formula can be equivalently rewritten as a conjunction of universal

formulas asserting the fact that a polyhedron P is contained in a union of polyhedra P_1, \dots, P_n . So this problem class is also coNP-complete [Srivastava 1993].

3. Linear diophantine constraints

In this section, we study linear arithmetic constraints over the integer numbers. While linear equations over \mathbb{Z} are still solvable in polynomial time, see Section 3.1, linear inequalities over \mathbb{Z} (and also linear equations over \mathbb{N}) are NP-hard, see Sections 3.2 to 3.7. A lot of work on linear diophantine constraints has been done in integer linear programming. Standard references are [Schrijver 1986, Nemhauser and Wolsey 1988, Wolsey 1998, Hooker 2000].

3.1. Equations over \mathbb{Z}

3.1.1. One linear equation over \mathbb{Z}

A linear diophantine equation

$$\alpha_1 x_1 + \dots + \alpha_n x_n = \beta, \quad (3.1)$$

with $\alpha_1, \dots, \alpha_n, \beta \in \mathbb{Z}$ has an integer solution if and only if the greatest common divisor $\gcd(\alpha_1, \dots, \alpha_n)$ of the coefficients on the left-hand side divides the right-hand side β . Using the Euclidean algorithm, we can decide in polynomial time if a solution exists, and if so, find one. We start by computing $\alpha, \gamma, \delta \in \mathbb{Z}$ such that

$$\alpha' = \gcd(\alpha_1, \alpha_2) = \alpha_1 \gamma + \alpha_2 \delta. \quad (3.2)$$

Then we solve recursively the linear diophantine equation in $n - 1$ variables

$$\alpha' x' + \alpha_3 x_3 + \dots + \alpha_n x_n = \beta. \quad (3.3)$$

If (3.3) has no solution, then (3.1) has no solution. If (3.3) has the solution x', x_3, \dots, x_n , then $x_1 = \gamma x', x_2 = \delta x', x_3, \dots, x_n$ defines an integral solution to (3.1).

3.1.2. Systems of linear diophantine equations over \mathbb{Z}

Next we consider systems of linear equations over \mathbb{Z} . There are the following basic results.

3.1. PROPOSITION (Solvability). *Given a rational matrix $A \in \mathbb{Q}^{m \times n}$ and a rational vector $b \in \mathbb{Q}^m$, the system $Ax = b$ has an integral solution $x \in \mathbb{Z}^n$ if and only if $y^T b$ is an integer, for each vector $y \in \mathbb{Q}^m$ for which $y^T A$ is integral.*

3.2. THEOREM. *Given a system $Ax = b$ with $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m$ we can decide in polynomial time if it has an integral solution $x \in \mathbb{Z}^n$, and if so, find*

one. If the system is solvable, we can also find in polynomial time integral vectors $x^0, x^1, \dots, x^k \in \mathbb{Z}^n$ such that

$$\text{Sol}_{\mathbb{Z}}(Ax = b) = x^0 + \mathbb{Z} \cdot x^1 + \dots + \mathbb{Z} \cdot x^k, \quad (3.4)$$

with x^1, \dots, x^k linearly independent.

Theorem 3.2 can be proved by computing the Hermite normal form of the matrix A . We say that a rational matrix A of full row rank is in *Hermite normal form* [Hermite 1851] if it has the form $(B \ 0)$, where B is a non-singular, lower triangular, non-negative matrix in which each row has a unique maximal entry, and this entry lies on the maximal diagonal of B . Hermite mentioned that every rational matrix A of full row rank can be brought into Hermite normal form by a series of *elementary integer column operations* of the form

- exchanging two columns
- multiplying a column by -1
- adding an integral multiple of one column to another column.

The Hermite normal form of A is uniquely determined by A . The classical Hermite normal form algorithm is an integer variant of Gaussian elimination. Although the encoding length of the Hermite normal form $(B \ 0)$ of the rational matrix A (of full row rank) is bounded by a polynomial in $\langle A \rangle$, and the number of elementary operations is also polynomially bounded, the intermediate numbers calculated during the execution can become enormously large. Hafner and McCurley [1991] give an example of a 20×20 integer matrix with entries only between 0 and 10, but which needs integers of up to 1500 decimal digits in a standard Hermite normal form algorithm. Fang and Havas [1997] proved an exponential lower bound on the encoding length of the intermediate entries in a well-defined variant of integer Gaussian elimination for Hermite normal form computation.

The first polynomial algorithm for computing the Hermite normal form was found by Frumkin [1976], who uses modular arithmetic to control the size of the entries that arise during the transformation. Others that have used modular arithmetic in various forms include [Schrijver 1986, Dormich, Kannan and Trotter 1987, Iliopoulos 1989a, Iliopoulos 1989b, Hafner and McCurley 1991, Storjohann and Labahn 1996]. Let $A \in \mathbb{Z}^{m \times n}$ be an integer matrix of rank m . Let $\delta = |\det(A)|$ be the absolute value of the determinant of some submatrix of A of rank m and consider the $m \times (n+m)$ -matrix $A' = (A \mid D)$ obtained from A by extending it with the $m \times m$ diagonal matrix $D = \text{diag}(\delta, \dots, \delta)$. One easily checks that the Hermite normal forms of A and A' are the same except for the last m columns of the Hermite normal form of A' , which are all zero. Therefore, we may work with A' instead of A . The basic idea to prevent explosion of the intermediate entries is then to perform further elementary operations using the m additional columns in A' to bring all entries in the original n columns between 0 and δ .

Kannan and Bachem [1979] used a rearrangement in the order of operations of the classical Hermite normal form algorithm and proved a polynomial time and space bound. In contrast to previous approaches that built the Hermite normal

form row by row, their algorithm successively puts the submatrices consisting of the first i row and columns into Hermite normal form. Chou and Collins [1982] improved Kannan and Bachem's procedure and gave a better space bound. Havas, Holt and Rees [1993] and Havas and Majewski [1994] developed a heuristic pivot selection strategy to reduce the intermediate entry growth in Gaussian elimination methods, for which they obtain good empirical results.

Smith [1861] proved that any integer matrix A can be transformed by elementary integer row and column operations into the *Smith normal form*

$$\begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}, \quad (3.5)$$

where $D = \text{diag}(\delta_1, \dots, \delta_k)$ and $\delta_1, \dots, \delta_k$ are positive integers such that δ_i divides δ_j , for all $1 \leq i \leq j \leq k$. Computing the Smith normal form is studied, for example, in [Storjohann 1997, Havas and Majewski 1997].

3.1.3. Lattice reduction

A lattice in \mathbb{R}^m is any set of the form

$$L = L(a_1, \dots, a_n) = \mathbb{Z}a_1 + \dots + \mathbb{Z}a_n, \quad (3.6)$$

with $a_1, \dots, a_n \in \mathbb{R}^m$. A system of linear diophantine equations $Ax = b$ is solvable over \mathbb{Z}^n iff the right-hand side b belongs to the lattice in \mathbb{R}^m that is generated by the columns of A . The solutions of a homogeneous system $Ax = 0$, $x \in \mathbb{Z}^n$ form a lattice in the space \mathbb{R}^n .

If the generators a_1, \dots, a_n of a lattice L are linearly independent, then (a_1, \dots, a_n) is called a *basis* of the lattice. Clearly, a lattice L may have several bases. Among the many possible bases, one would like to select one with some nice properties. The famous *LLL algorithm* [Lenstra, Lenstra and Lovász 1982] computes for given $a_1, \dots, a_n \in \mathbb{Q}^n$ in polynomial time a so-called *reduced* basis of the lattice $L(a_1, \dots, a_n)$. The purpose of a reduced basis is to have short vectors that are nearly orthogonal. This algorithm has many important applications, e.g. in diophantine approximation or in cryptography. It can also be used to compute the Hermite normal form of an integer matrix [Havas, Majewski and Matthwes 1998].

The LLL algorithm computes in polynomial time an approximation of a shortest vector in the lattice. The problem of finding a nonzero lattice vector $v \in L(a_1, \dots, a_n)$ such that the norm $\|v\|$ is minimal is called the *shortest lattice vector problem*. For the maximum norm $\|x\|_\infty = \max_{i=1, \dots, m} |x_i|$, this problem is known to be NP-hard [van Emde Boas 1981]. The question whether this problem is NP-hard also for the Euclidean norm $\|x\|_2 = \sqrt{x^T x}$ has been open for almost two decades. For randomized reductions, it was recently solved in the affirmative by Ajtai [1998]. A related, but essentially different problem is the *nearest lattice vector problem*. It consists in finding for a vector $b \in \mathbb{Q}^n$ a vector $v \in L(a_1, \dots, a_n)$ such that $\|b - v\|$ is minimal. This problem is known to be NP-hard for any norm [van Emde Boas 1981]. Henk [1997] showed that the shortest lattice vector problem is

polynomially reducible to the nearest lattice vector problem. Various authors have studied recently the complexity of approximating shortest or nearest lattice vectors, see e.g. [Arora, Babai, Stern and Sweedyk 1993, Blömer and Seifert 1999]. The *minimum basis problem* of finding a basis (b_1, \dots, b_n) such that $\|b_1\| \cdot \dots \cdot \|b_n\|$ is minimal, is also NP-hard [Grötschel et al. 1988].

3.2. Equations over \mathbb{N} , inequalities over \mathbb{Z}

All constraint problems that we have considered so far can be solved in polynomial time. In this section, we consider systems of linear inequalities over the integer numbers

$$Ax \leq b, x \in \mathbb{Z}^n, \quad (3.7)$$

and, as a special case, systems of linear equations over the non-negative integer numbers

$$Ax = b, x \in \mathbb{N}^n. \quad (3.8)$$

Any set of *propositional clauses* of the form

$$x_1 \vee \dots \vee x_m \vee \bar{y}_1 \vee \dots \vee \bar{y}_k \quad (3.9)$$

can be translated into a system of *clausal inequalities* of the form

$$\begin{array}{rcll} x_1 + \dots + x_m + (1 - y_1) + \dots + (1 - y_k) & \geq & 1 & \text{or} \\ x_1 + \dots + x_m - y_1 - \dots - y_k & \geq & 1 - k. & \end{array} \quad (3.10)$$

The clause set is satisfiable if and only if the corresponding system of clausal inequalities together with the bounds $0 \leq x_i, y_j \leq 1$ has an integer solution. This implies immediately that deciding the satisfiability of a system of linear diophantine inequalities is NP-hard.

The solution set of (3.7) may be infinite. However, there is always a finite parametric description.

3.3. THEOREM. *Let $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$, with $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$, be a rational polyhedron and let $S = P \cap \mathbb{Z}^n$ be the set of integer points contained in P . Then there exist finitely many points p^1, \dots, p^k in S and a finitely many rays of P $r^1, \dots, r^l \in \mathbb{Z}^n$ such that*

$$S = \left\{ \sum_{i=1}^k \lambda_i p^i + \sum_{j=1}^l \mu_j r^j \mid \lambda_i, \mu_j \in \mathbb{N}, \sum_{i=1}^k \lambda_i = 1 \right\}. \quad (3.11)$$

The absolute value of all p^i and r^j can be bounded by $(n+1)\Delta$, where Δ is the maximum absolute value of the subdeterminants of the matrix $(A \mid b)$.

In particular, this theorem implies that if a system of linear inequalities has an integral solution, then it has one with encoding length polynomially bounded by $\langle (A \mid b) \rangle$. Therefore, we can conclude:

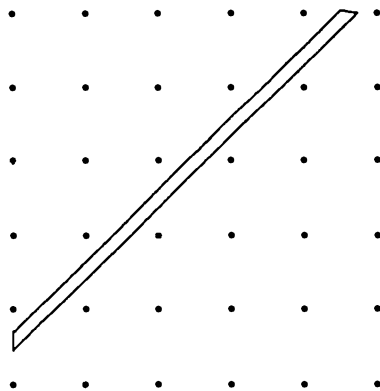


Figure 6: Polytope with empty integer hull

3.4. THEOREM. *The problem of deciding for a rational matrix $A \in \mathbb{Q}^{m \times n}$ and a rational vector $b \in \mathbb{Q}^m$ whether the system of linear inequalities $Ax \leq b$ has an integral solution $x \in \mathbb{Z}^n$ is NP-complete.*

In contrast to the polynomial solvability of 2SAT, the problem stays NP-complete if we admit only two variables in each constraint [Lagarias 1985]. The following problems are also NP-complete:

- Given $A \in \mathbb{Q}^{n \times n}$, $b \in \mathbb{Q}^n$, does $Ax = b$ have a *nonnegative* integral solution $x \in \mathbb{N}^n$?
- Given $A \in \mathbb{Q}^{n \times n}$, $b \in \mathbb{Q}^n$, does $Ax = b$ have a 0-1 solution $x \in \{0, 1\}^n$?
- Given $a \in \mathbb{Q}^n$, $\beta \in \mathbb{Q}$, does $a^T x = \beta$ have a *nonnegative* integral solution $x \in \mathbb{N}^n$?
- Given $a \in \mathbb{Q}^n$, $\beta \in \mathbb{Q}$, does $a^T x = \beta$ have a 0-1 solution $x \in \{0, 1\}^n$?
- Knapsack problem: Given $a, c \in \mathbb{Q}^n$ and $\beta, \delta \in \mathbb{Q}$, is there a 0-1 vector $x \in \{0, 1\}^n$ with $a^T x \leq \beta$ and $c^T x \geq \delta$?

While solving linear diophantine inequalities in general is NP-hard, a celebrated result of Lenstra [1983] states that this can be done in polynomial time, if the dimension n is fixed. Before Lenstra's theorem was obtained, this was known only for the cases $n = 1, 2$.

3.5. THEOREM (Lenstra 83). *For each fixed natural number n , there exists a polynomial algorithm which finds an integral solution $x \in \mathbb{Z}^n$ of a given system $Ax \leq b$, $A \in \mathbb{Q}^{n \times n}$, $b \in \mathbb{Q}^n$ in n variables, or decides that no such solution exists.*

The geometric intuition underlying Lenstra's algorithm can be described as follows [Aardal, Weismantel and Wolsey 1999, Aardal, Hurkens and Lenstra 1998a]. Consider a bounded polyhedron P like the one in Fig. 6, which may extend arbitrarily far in one direction. Depending on the length of P , a standard branch-and-bound

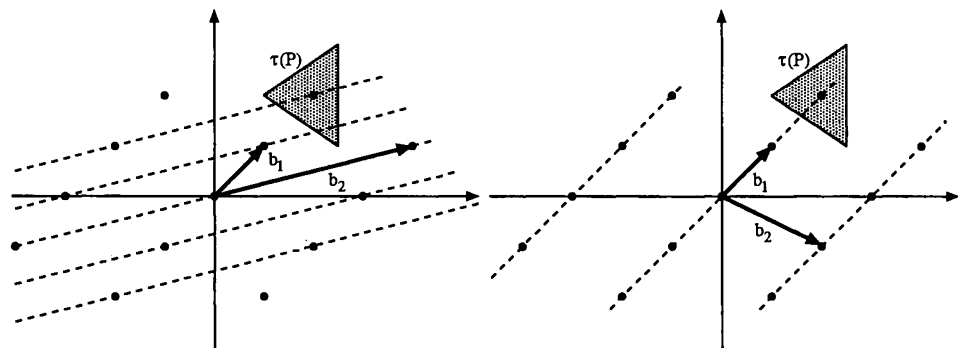


Figure 7: Non-orthogonal vs. nearly orthogonal lattice bases

algorithm (see Section 3.4) may require arbitrarily many iterations before concluding that no feasible solution exists. Suppose the polytope P is full-dimensional and consider the lattice \mathbb{Z}^n with the lattice basis e_1, \dots, e_n . Our problem is to decide whether there exists a point $x \in P \cap \mathbb{Z}^n$. To avoid working with the thin polytope P , we may apply a linear transformation τ to make it appear more regular. Then we have to decide whether there exists a point $y \in \tau(P) \cap \tau(\mathbb{Z}^n)$. The new polytope $\tau(P)$ has a regular shape, but the basis vectors $\tau(e_j)$ are not necessarily orthogonal any longer, so the difficulty in branching is still present. To overcome this situation, we may use lattice reduction, see Section 3.1.3, to obtain a new basis (b_1, \dots, b_n) of the lattice $\tau(\mathbb{Z}^n)$ having short and nearly-orthogonal vectors. In particular, one can show that the distance between two consecutive hyperplanes $H + kb_n, H + (k+1)b_n$, where $H = \mathbb{R}b_1 + \dots + \mathbb{R}b_{n-1}$ and $k \in \mathbb{Z}$, is not too short, which means that if we branch on these hyperplanes, then there cannot be too many of them, see Fig. 7.

Lovász and Scarf [1992] developed a *generalized basis reduction algorithm* for integer linear optimization. Like Lenstra's approach, this algorithm uses branching on hyperplanes. However, instead of using the transformation τ to transform the polytope and the initial basis, and then applying the standard basis reduction, their algorithm measures the width of the polytope in different independent directions to produce a so-called *Lovász-Scarf reduced basis*. Cook, Rutherford, Scarf and Shallcross [1993] describe a successful implementation of this algorithm.

Aardal et al. [1998a, 1998b] develop an algorithm for solving systems of linear diophantine equations with lower and upper bounds on the variables that is based on lattice basis reduction. This algorithm first finds a short vector satisfying the system of linear diophantine equations, and a set of vectors solving the corresponding homogeneous system. Due to basis reduction, all these vectors are relatively short. The next step is to branch on linear combinations of the solutions of the homogeneous system, which either yields a vector that satisfies the bound constraints or provides a proof that no such vector exists.

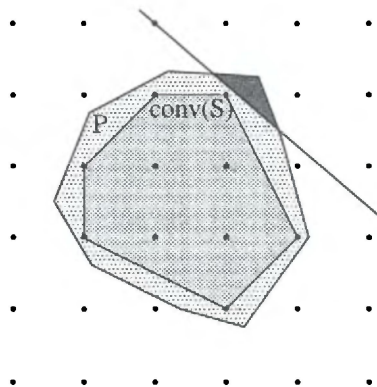


Figure 8: Cutting plane

3.3. Cutting planes

Consider the *integer linear optimization* or *integer linear programming* problem

$$\max\{c^T x \mid Ax \leq b, x \in \mathbb{Z}^n\}, \quad A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m. \quad (3.12)$$

An important consequence of Theorem (3.3) is that for a rational polyhedron $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$, the convex hull of the set of integer points $S = P \cap \mathbb{Z}^n$ in P is again a rational polyhedron. Therefore, in principle, the integer linear optimization problem (3.12) can be reduced to an ordinary linear optimization problem

$$\max\{c^T x \mid x \in \text{conv}(S) \subseteq \mathbb{F}^n\} \quad (3.13)$$

over the rational polyhedron $\text{conv}(S)$. If (3.13) has a bounded optimal value, then it has an optimal solution, namely an extreme point of $\text{conv}(S)$, which is an optimal solution of (3.12). The objective value of (3.12) is unbounded if and only if the objective value of (3.13) is unbounded.

In order to solve the linear optimization problem (3.13), we need a linear inequality system $Cx \leq d$ defining the polyhedron $\text{conv}(S)$. In general, the polyhedron $\text{conv}(S)$ will have exponentially many facets. Therefore, such a description cannot be obtained in polynomial time. However, in order to find an optimal solution of (3.12), an approximation of $\text{conv}(S)$ might be sufficient. This idea leads to *cutting plane algorithms* for solving the integer linear optimization problem. We start with the *linear programming relaxation* $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$ of the original problem. Then we add new inequalities which are satisfied by all points in S but which cut off at least one fractional vertex of P . These are called *cutting planes* (Fig. 8). This is repeated until an integer vertex is obtained, which belongs to the integer solution set S and maximizes $c^T x$.

3.3.1. Gomory-Chvátal cutting planes

One basic inference rule for deriving cutting planes is the Gomory-Chvátal rounding principle:

$$\text{Rounding: } \frac{a^T x \leq \beta}{a^T x \leq \lfloor \beta \rfloor} \text{ if } a \in \mathbb{Z}^n \quad (3.14)$$

If $a \in \mathbb{Z}^n$ is integer, then for all integer vectors $x \in \mathbb{Z}^n$, the left-hand side of the inequality $a^T x \leq \beta$ is integer. Therefore, we do not lose any integer solution if we round down the right-hand side β to the next integer number $\lfloor \beta \rfloor$. Geometrically, this means the following. Assume that the entries of a are coprime integer numbers and that β is fractional. Then rounding β means that we translate the hyperplane $a^T x = \beta$ until it meets an integer point. Given a system of linear inequalities $Ax \leq b$, a *Gomory-Chvátal cutting plane* is obtained by combining the rules `Nonneg_lin_com` and `Rounding`:

$$\text{GC_cp: } \frac{Ax \leq b}{u^T Ax \leq \lfloor u^T b \rfloor} \text{ if } \begin{cases} u \geq 0, \\ u^T A \in \mathbb{Z}^n \end{cases} \quad (3.15)$$

A *cutting plane proof* of length l of an inequality $c^T x \leq \delta$ from $Ax \leq b$ is a sequence of inequalities $c_1^T x \leq \delta_1, \dots, c_l^T x \leq \delta_l$ such that

$$\text{CG_cp: } \frac{Ax \leq b, c_1^T x \leq \delta_1, \dots, c_{i-1}^T x \leq \delta_{i-1}}{c_i^T x \leq \delta_i} \quad (3.16)$$

for each $i = 1, \dots, l$, and

$$\text{weak_rhs: } \frac{c_l^T x \leq \delta_l}{c^T x \leq \delta}. \quad (3.17)$$

Gomory-Chvátal cutting planes provide a complete inference system for linear diophantine inequalities.

3.6. THEOREM (Chvátal 73, Schrijver 80). *Let $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$ be a non-empty polyhedron that is rational or bounded and let $S = P \cap \mathbb{Z}^n$.*

- *If $S \neq \emptyset$ and $c^T x \leq \delta$ is valid for S , then there is a cutting plane proof of $c^T x \leq \delta$ from $Ax \leq b$.*
- *If $S = \emptyset$, then there is a cutting plane proof starting from $Ax \leq b$ yielding the contradiction $0 \leq -1$.*

In the given form, the Gomory-Chvátal procedure is not effective. It is not clear how to choose the weights in the non-negative linear combination of the given constraint matrix. However, several well-known inference rules are subsumed by the Gomory-Chvátal principle, see Section 3.7. The first cutting plane procedure for solving integer linear optimization problems was proposed by Gomory in 1958. Here, the cutting planes are generated from a basic feasible solution in the Simplex algorithm. If v is a vertex of the polyhedron P with some fractional component, then by a simple rounding operation one can obtain a valid inequality for $P \cap \mathbb{Z}^n$ that cuts off v . Gomory showed that by systematically adding these cuts, and using the dual Simplex method with appropriate anticycling rules, one can obtain a finitely

terminating cutting plane algorithm for general integer linear optimization problems. The practical performance of Gomory's original cutting plane algorithm is not good. However, when used within a branch-and-cut framework (see Section 3.4.3), Gomory's cutting planes turned to be very useful [Balas, Ceria, Cornuéjols and Natraj 1996]. As a consequence, they are currently used in most state-of-the-art integer programming software packages.

The set of vectors P' satisfying all cutting planes for P is called the *elementary closure* of P . Let $P^{(0)} = P$ and $P^{(i+1)} = (P^{(i)})'$, for $i \geq 0$. The *Chvátal rank* of P is the smallest number t such that $P^{(t)} = P_I$, where $P_I = \text{conv}(P \cap \mathbb{Z}^n)$ is the *integer hull* of P . It indicates the "degree of discreteness" of the underlying integer linear optimization problem. Chvátal [1973] showed that every bounded polyhedron $P \subseteq \mathbb{R}^n$ has finite rank. Schrijver [1980] extended this result to possibly unbounded, but rational polyhedra $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$, where $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$. Cook, Gerards, Schrijver and Tardos [1986] and Gerards [1990] proved that for every matrix $A \in \mathbb{Z}^{m \times n}$ there exists $t \in \mathbb{N}$ such that for all right-hand sides $b \in \mathbb{Z}^m$, the Chvátal rank of $P_b = \{x \in \mathbb{F}^n \mid Ax \leq b\}$ is bounded by t . Already in dimension 2, there exist rational polyhedra of arbitrarily large Chvátal rank [Chvátal 1973]. Bockmayr, Eisenbrand, Hartmann and Schulz [1999] and Eisenbrand and Schulz [1999] studied the Chvátal rank of polytopes contained in the n -dimensional 0/1 cube $[0, 1]^n$. They showed that the Chvátal rank of a polytope $P \subseteq [0, 1]^n$ is $O(n^2 \log n)$ and prove the linear upper and lower bound n for the case $P \cap \mathbb{Z}^n = \emptyset$. Eisenbrand [1999] proved that the problem

Given $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$ and a rational vector $\hat{x} \in \mathbb{Q}^n$, decide whether \hat{x} is outside the elementary closure P' of the polyhedron $P = \{x \in \mathbb{F}^n \mid Ax \leq b\}$.

is NP-complete. In other words, it is NP-complete to decide whether there is an inference using rule CG_cp that separates the point \hat{x} from P' .

Caprara, Fischetti and Letchford [2000] apply Gaussian elimination in the field $\mathbb{Z}/k\mathbb{Z}$, k a prime number, to compute for a given *amount of violation* $\nu \in \{0, \dots, k-1\}$ a Gomory-Chvátal cutting plane $u^T Ax \leq \lfloor u^T b \rfloor$, with $u^T A \in k\mathbb{Z}$ and $u^T b - \lfloor u^T b \rfloor = \nu/k$. Bockmayr and Eisenbrand [1999] use Hermite normal form computation in the residue ring $\mathbb{Z}/d\mathbb{Z}$, where $d = |\det(A)|$, to compute in polynomial time a Gomory-Chvátal cutting plane with maximal amount of violation. They also derive, in fixed dimension, a polynomial upper bound on the number of inequalities needed to describe the elementary closure.

3.3.2. Disjunctive cutting planes

A second principle to derive cutting planes is based on disjunctive programming [Balas 1979, Balas 1998, Sherali and Shetty 1980]. A *disjunctive optimization problem* is of the form

$$\max\{c^T x \mid \bigvee_{h \in H} A^h x \leq b^h, x \geq 0\}, \quad (3.18)$$

for H finite, $A^h \in \mathbb{F}^{m_h \times n}$, and $b^h \in \mathbb{F}^{m_h}$. The feasible set of the disjunctive optimization problem (3.18) is the union of the polyhedra $P^h = \{x \in \mathbb{F}^n \mid A^h x \leq$

$b^h, x \geq 0\}$, $h \in H$. *Disjunctive cutting planes* are obtained by applying the rule

$$\text{disj_cp} : \frac{\bigvee_{h \in H} A^h x \leq b^h, x \geq 0}{a^T x \leq \beta} \quad \text{if} \quad \begin{cases} u_h \in \mathbb{R}^{m_h}, u_h \geq 0, \\ a^T \leq u_h^T A^h, \\ u_h^T b^h \leq \beta, \\ \text{for all } h \in H. \end{cases} \quad (3.19)$$

We first take a non-negative linear combination of each disjunct and then weaken the left-hand and the right-hand side of all the inequalities obtained. The constraint $x \geq 0$ allows us to weaken also the left-hand side. The completeness of the disjunctive cutting plane principle was studied by several authors, see e.g. [Blair and Jeroslow 1978, Jeroslow 1977]. To apply the inference rule *disj_cp* to integer programming problems, one may, e.g., consider disjunctions of the form

$$\begin{array}{ccc} Ax \leq b, x \geq 0 & \bigvee & Ax \leq b, x \geq 0 \\ x_j \leq d & & x_j \geq d + 1, \end{array} \quad (3.20)$$

for some $j \in \{1, \dots, n\}$ and $d \in \mathbb{Z}$. A successful branch-and-cut algorithm for general linear 0-1 optimization, which is based on disjunctive cutting planes, is the *lift-and-project method* developed by Balas, Ceria and Cornuéjols [1993, 1996]. Recently, some interesting connections between disjunctive cuts and Gomory's mixed integer cut have been discovered [Balas and Perregaard 2000].

3.4. Branch-and-infer

Although there exist complete inference systems for solving linear diophantine inequalities, like the Gomory-Chvátal rounding procedure, pure cutting plane algorithms do not work well in practice. From the practical point of view, it is much more efficient to combine inference with some kind of search. We next develop a general framework, *branch-and-infer* [Bockmayr and Kasper 1998, Kasper 1998], which allows us to describe various approaches for solving constraint problems over the integers in a uniform way. In particular, this framework unifies classical *integer linear programming* [Nemhauser and Wolsey 1988, Wolsey 1998, Johnson, Nemhauser and Savelsbergh 2000] and *finite domain constraint programming* [van Hentenryck 1989, Jaffar and Maher 1994, van Hentenryck and Saraswat 1996, Wallace 1996, Marriott and Stuckey 1998].

Branch-and-infer is based on a distinction between *primitive* and *non-primitive constraints*. Intuitively, the primitive constraints are those constraints that can be solved easily. The non-primitive constraints are the difficult constraints, which make the problem hard. In integer linear programming (ILP), the primitive constraints are linear equations and inequalities when solved over the real (or rational) numbers. The only non-primitive constraint is *integer*(x), i.e. the condition that certain variables should take integer values:

$$\begin{aligned} \text{Prim(ILP)} &= \{a^T x \diamond \beta \mid a \in \mathbb{Z}^n, \beta \in \mathbb{Z}, \diamond \in \{\leq, \geq, =\}\} \\ \text{NPrim(ILP)} &= \{\text{integer}(x)\} \end{aligned} \quad (3.21)$$

In finite domain constraint programming (CP(FD)), the primitive constraints either restrict the domain of one variable, or express equations between two variables. All other constraints are non-primitive. This includes linear equations, inequalities or disequalities in several variables, and symbolic constraints like `alldifferent(x)`, which states that certain variables should take pairwise different values:

$$\begin{aligned} \text{Prim}(FD) &= \{x_i \diamond \alpha, x_i = x_j \mid \alpha \in \mathbb{Z}, \diamond \in \{\leq, \geq, =, \neq, >, <\}\} \\ &\quad \cup \{\text{integer}(x)\} \\ \text{NPrim}(FD) &= \{a^T x \diamond \beta \mid a \in \mathbb{Z}^n, \beta \in \mathbb{Z}, \diamond \in \{\leq, \geq, =, \neq, >, <\}\} \\ &\quad \cup \{\text{alldifferent}(x), \dots\} \end{aligned} \quad (3.22)$$

The primitive constraints are collected in the *constraint store* [Saraswat 1993]. They define a *relaxation* of the problem for which an efficient solution method is available. The non-primitive constraints are handled locally by an *inference agent* that derives from a given non-primitive constraint and the current relaxation new primitive constraints that tighten this relaxation. Since, in general, a problem cannot be solved using the relaxation alone, inference has to be combined with search, which together provide a complete solution method. To describe branch-and-infer in a formal way, we use transition rules of the form

$$- : \frac{\langle P, S \rangle}{\langle P', S' \rangle} \text{ if } \text{Cond} \quad (3.23)$$

saying that from a computation state $\langle P, S \rangle$ we may proceed to a computation state $\langle P', S' \rangle$ if the conditions in *Cond* are satisfied. Here $P = \{C_1, \dots, C_m\}$ (resp. P') denotes a set of constraint (sub-)problems C_1, \dots, C_m , which logically corresponds to the disjunction $C_1 \vee \dots \vee C_m$. The set S (resp. S') denotes a set of feasible solutions. For a set T and an element t , we will write $t.T$ instead of $\{t\} \cup T$. When solving a constraint problem C , the initial state is $\langle \{C\}, \emptyset \rangle$, and the final state is $\langle \emptyset, \{S\} \rangle$. If $S = \emptyset$, then the problem is infeasible.

The operational behaviour of the non-primitive constraints is formalized by the rule

$$\text{bi_infer} : \frac{\langle (c \wedge C).P, S \rangle}{\langle (p \wedge (c \wedge C)).P, S \rangle} \text{ if } \begin{cases} c \text{ is non-primitive,} \\ p \text{ is primitive,} \\ \text{Prim}(C) \wedge c \rightarrow p, \\ \text{Prim}(C) \not\vdash p. \end{cases} \quad (3.24)$$

In integer linear programming, the relaxation obtained from the primitive constraints is the standard linear programming relaxation. Inferring a new primitive constraint corresponds to the generation of a cutting plane that cuts off some part of this relaxation. In finite domain constraint programming, the basic inference principle is *domain reduction*. The corresponding *local consistency* techniques have been studied in artificial intelligence for a long time, see e.g. [Mackworth 1977, Tsang 1993]. For each non-primitive constraint, so-called *propagation* algorithms try to remove inconsistent values from the domain of the variables occurring in the constraint. From a logical point of view, this corresponds to the

generation of a new primitive constraint of the form $x \leq u, x \geq l$, which is called *bound reasoning*, or $x \neq v$, which is called *domain reasoning*. Whenever the domain of a variable changes, all propagation algorithms of the constraints in which this variable occurs may become active and further reduce the domains of their variables.

In general, inferring primitive constraints alone will be not sufficient to solve the given problem, either because the primitive constraints are not expressive enough, like in CP(FD), or because exponentially many of them would be necessary, like in ILP. Therefore, we need a second principle that allows us to tighten the relaxation further if the inference process has been suspended. The idea is to split the problem into subproblems and to process each subproblem independently of the others. The subproblems are obtained by setting up branching constraints and adding to each of them one copy of the problem under consideration. If the branching constraints are chosen in the right way, the relaxation of a subproblem will be strictly stronger than the relaxation of the father problem. Therefore, the inference agents associated with the non-primitive constraints may become active again and derive new primitive constraints. The branching operation is described by the rule

$$\text{bi_branch} : \frac{\langle C.P, S \rangle}{\langle \{c_1 \wedge C, \dots, c_k \wedge C\} \cup P, S \rangle} \text{ if } \begin{cases} C \equiv C \wedge (\bigvee_{i=1}^k c_i), \\ c_i \text{ is primitive,} \\ \text{Prim}(C) \not\vdash c_i, \\ i = 1, \dots, k. \end{cases} \quad (3.25)$$

The constraints c_1, \dots, c_k are called the *branching constraints*, the problems $\{c_1 \wedge C, \dots, c_k \wedge C\}$ are the new *subproblems*. In many applications, we will have a binary branching of the form $c_1 \equiv c, c_2 \equiv \neg c$. By repeated application of the branching rule, we build up a *search tree*. Theoretically, we can get a complete enumeration of all the solutions in $\text{Sol}_{\mathbb{Z}}(C)$. In practice, generating new subproblems has to be avoided as much as possible. Branching can be avoided if the (sub-)problem is infeasible. Since deciding the satisfiability of the whole problem is intractable, we test feasibility only on the primitive constraints, i.e. the relaxation. The next rule describes pruning by the infeasibility of the relaxation, which is denoted by \perp .

$$\text{bi_clash} : \frac{\langle C.P, S \rangle}{\langle P, S \rangle} \text{ if } \text{Prim}(C) \rightarrow \perp \quad (3.26)$$

The transition rules *bi_infer*, *bi_branch* and *bi_clash* are the basic rules in the branch-and-infer framework. What is still missing are rules that describe when a solution has been obtained. This depends on the type of problem to be solved, i.e. whether we want to find feasible or optimal solutions.

Solving a *constraint satisfaction* problem means deciding whether the problem is satisfiable and if so computing one or more feasible solutions. We hide the concrete method of computing feasible solutions from the relaxation and the way they are represented in a function *extract*. This function has to be chosen properly according to whether one wants to compute only one solution or more. This yields our

next rule:

$$\text{bi_sol} : \frac{\langle C.P, S \rangle}{\langle P, S \cup S^* \rangle} \text{ if } \begin{cases} S^* = \text{extract}(\text{Prim}(C)) \\ S^* \rightarrow C. \end{cases} \quad (3.27)$$

If one wants to compute more or all solutions, then repeated application of this rule to the different subproblems will collect the different solution families. Thus the task of finding more solutions is left to the control strategy for the application of the different transition rules.

In *constrained optimization*, one would like to compute a feasible solution that is optimal with respect to some objective function. For a maximization problem $\max\{f(x) \mid x \in \text{Sol}_Z(C)\}$, feasible solutions in $\text{Sol}_Z(C)$ yield lower bounds for the maximum value of f . In order to find an optimal solution, there exist two general methods, branch-and-bound, as it is used in finite domain constraint programming, and branch-and-relax resp. branch-and-cut, which are standard techniques in integer linear programming. Note that we follow here the terminology of constraint programming, where branch-and-relax corresponds to what is usually called branch-and-bound in integer linear programming. We now formalize the different approaches in our framework.

3.4.1. Branch-and-bound

The *branch-and-bound* method is characterized by using only lower bounds to find an optimal solution. Thus we solve a sequence of satisfiability problems leading successively to better solutions. More precisely, we repeatedly compute a feasible solution $s^* \in \text{Sol}_Z(C)$ and then add the constraint $f(x) \geq f(s^*) + 1$ to all the subproblems of our search tree, which restricts the set of feasible solutions to those that yield better objective function values. The constraint $f(x) \geq f(s^*) + 1$ is called a *lower bounding constraint*. If after adding a lower bounding constraint, all the subproblems become infeasible, then the last feasible solution is optimal. We require that f takes always integral values if x is integral since otherwise feasible solutions may be lost and the global optimum cannot be found. To describe branch-and-bound in our framework, we extend the transition system consisting of the rules *bi.branch*, *bi.clash* by the rule

$$\text{bi_climb} : \frac{\langle \{C_1, \dots, C_n\}, \{s\} \rangle}{\langle \{c \wedge C_1, \dots, c \wedge C_n\}, \{s^*\} \rangle} \text{ if } \begin{cases} s^* = \text{extract}(\text{Prim}(C_1)) \\ f(s^*) > f(s) \\ c \equiv (f(x) \geq f(s^*) + 1). \end{cases} \quad (3.28)$$

Here, the function *extract* is again responsible for computing a feasible solution of the relaxation. If no feasible solution is known, we assume $f(s) = -\infty$.

3.4.2. Branch-and-relax

In contrast to branch-and-bound, which uses only lower bounds, *branch-and-relax* works with two bounds. In addition to the global lower bound *glb* obtained from a feasible solution, we compute for each subproblem a local upper bound *lub*. For example, this can be done by optimizing the objective function subject to the

relaxation of a subproblem, i.e. the primitive constraints in the constraint store. We describe branch-and-relax again by an extension of the transition system given by the rules `bi_clash`, `bi_branch`. The local upper bounds allow us to introduce a new rule to prune the search tree. If a local upper bound is smaller than the best known global lower bound, then the corresponding subproblem cannot lead to a better solution and therefore can be discarded.

$$\text{bi_bound} : \frac{\langle C.P, \{s\} \rangle}{\langle P, \{s\} \rangle} \text{ if } \max\{f(x) \mid x \in \text{Sol}_Z(C)\} \leq \text{lub} \leq f(s) \quad (3.29)$$

Furthermore, when computing a local upper bound, we may find an optimal solution of a subproblem that yields a better feasible solution of the whole problem.

$$\text{bi_opt} : \frac{\langle C.P, \{s\} \rangle}{\langle P, \{s^*\} \rangle} \text{ if } \begin{cases} \max\{f(x) \mid x \in \text{Sol}_Z(\text{Prim}(C))\} = f(s^*) \\ s^* \in \text{Sol}_Z(C) \\ f(s^*) > f(s) \end{cases} \quad (3.30)$$

Branch-and-relax is obtained from branch-and-bound by replacing the rule `bi_climb` with the two rules `bi_bound` and `bi_opt`.

3.4.3. Branch-and-cut

To apply branch-and-relax in practice, we must be able to compute local upper bounds in a computationally feasible way. For example, this is possible in integer linear programming, where we can obtain an upper bound by solving the linear programming relaxation. We may even find a feasible solution of the whole problem, so that the rule `bi_opt` can be applied. In order to obtain good lower bounds, we need *tight* linear relaxations. Given two systems $Ax \leq b$ and $A'x \leq b'$ defining the same set of feasible solutions, we say that $Ax \leq b$ is *tighter* than $A'x \leq b'$ if $\{x \in \mathbb{F}^n \mid Ax \leq b\} \subseteq \{x \in \mathbb{F}^n \mid A'x \leq b'\}$. Tight descriptions can be obtained statically by adding redundant inequalities to the problem formulation or dynamically by computing cutting planes that cut off some part of the linear relaxation without eliminating feasible solutions. Combining branch-and-bound with cutting plane generation, which is called *branch-and-cut*, is one of the most powerful techniques in ILP [Johnson et al. 2000]. In our framework, branch-and-cut is captured by the rules `bi_infer`, `bi_branch`, `bi_clash`, `bi_bound`, and `bi_opt`. Here, the last four rules describe branch-and-relax, while the first rule `bi_infer` allows for the generation of cutting planes.

3.5. Gröbner bases in integer programming

Another approach to solve integer linear optimization problems that recently has received a lot of interest is based on Gröbner bases [Conti and Traverso 1991, Thomas 1995, Thomas 1998, Ziegler 1999]. Consider an integer linear program of the form

$$\min\{c^T u \mid Au = b, u \in \mathbb{N}^n\}, \quad (3.31)$$

with $A \in \mathbb{Z}^{m \times n}$ of rank m , $b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^n$. A vector $u^* \in \mathbb{N}^n$ is a feasible solution of (3.31) iff $Au^* = b$. Consider the congruence class \equiv_A on the monoid \mathbb{N}^n defined by

$$u \equiv_A v \text{ iff } Au = Av. \quad (3.32)$$

Then the set of all feasible solutions of (3.31) corresponds to the congruence class $\{u \in \mathbb{N}^n \mid u \equiv_A u^*\}$. We are looking for an element of this class with minimal cost.

The starting point of the Gröbner basis approach is to identify a vector $u = (u_1, \dots, u_n) \in \mathbb{N}^n$ with a *monomial* $x^u = x_1^{u_1} \cdot \dots \cdot x_n^{u_n}$ in the polynomial ring $\mathbb{Q}[x_1, \dots, x_n]$. Then one considers the *toric ideal*

$$I_A = \langle x^u - x^v \mid Au = Av, u, v \in \mathbb{N}^n \rangle \quad (3.33)$$

generated by all *binomials*, i.e. monomial differences $x^u - x^v$ for which $u \equiv_A v$. A Gröbner basis $G_{A, \succ}$ of the ideal I_A , for some term order \succ , defines a confluent and terminating reduction relation on $\mathbb{Q}[x_1, \dots, x_n]$ that solves the word problem for the congruence relation generated by I_A . For any binomial p , we can compute a normal form $NF_{G_{A, \succ}}(p)$, which will again be a binomial. We have

$$u \equiv_A v \text{ iff } x^u - x^v \in I_A \text{ iff } NF_{G_{A, \succ}}(x^u - x^v) = 0. \quad (3.34)$$

If we choose a term order \succ_c that is compatible with the objective function, i.e. $x^u \succ_c x^v$ for $c^T u > c^T v$, then we can obtain an optimal solution u^* from a feasible solution u° simply by normalizing x^{u° with G_{A, \succ_c} . This yields the following algorithm

1. Compute the reduced Gröbner basis $G_{A, c}$ of I_A w.r.t. \succ_c .
2. Compute the normal form x^{u^*} of x^{u° w.r.t. $G_{A, c}$. Then u^* is an optimal solution of (3.31).

Note that the Gröbner basis $G_{A, c}$ does not depend on the right-hand side b . In some sense, we are solving problem (3.31) for all right-hand sides. A key technical problem is to obtain a good starting basis of I_A that is needed to compute the Gröbner basis with Buchberger's algorithm.

3.6. Computing Hilbert bases

Next we present methods for computing *all* solutions of a system of linear diophantine constraints over the nonnegative integers. This problem is at the heart of various algorithms for equational unification, e.g. in associative, associative-commutative or distributive theories. Properties like liveness and reachability of Petri nets can also be verified by completely solving a system of linear equations over \mathbb{N} . The first complete method for solving a single homogeneous diophantine equation was proposed by Elliott [1903] and extended by MacMahon [1916] to the case of systems of homogeneous equations. The importance of this problem for unification theory and automated deduction, see e.g. [Bürckert, Herold, Kapur, Siekmann, Stickel, Tepp and Zhang 1988, Adi and Kirchner 1992], has stimulated a

lot of research during the last 20 years. Various new algorithms have been proposed for solving

- one linear diophantine equation [Huet and Lang 1978, Fortenbacher 1993, Guck-enbiehl and Herold 1985, Clausen and Fortenbacher 1989, Lankford 1989, Filgueiras and Tomás 1991, Filgueiras and Tomás 1993],
- systems of linear diophantine equations [Romeuf 1990, Abdulrab and Pécuchet 1989, Pottier 1991a, Domenjoud 1991, Contejean and Devie 1994, Tomás and Filgueiras 1997, Pasechnik 1998],
- systems of linear diophantine equations, inequalities, and possibly disequalities [Abdulrab and Maksimenko 1995, Domenjoud and Tomás 1995, Boudet and Comon 1996, Ajili and Contejean 1997, Ajili and Lock 1998].

Our presentation is based on [Tomás 1997, Ajili 1998], which also contain overviews of the existing algorithms and a more detailed description of some of them.

3.6.1. Representing the solution set.

It has been known for a long time [Gordan 1873, Hilbert 1890] that the set of non-negative integer solutions of a system of homogeneous linear diophantine equations

$$Ax = 0, x \in \mathbb{N}^n, \quad (3.35)$$

with $A \in \mathbb{Z}^{m \times n}$, is a finitely generated submonoid \mathcal{M} of \mathbb{N}^n . There exists a unique finite subset $\mathcal{H}(M)$, which is called the *Hilbert basis* of \mathcal{M} , such that each solution of (3.35) is a non-negative integer linear combination of elements in $\mathcal{H}(M)$. In contrast to Carathéodory's theorem in the real case, more than n elements of the Hilbert basis might be necessary to represent a given solution $x \in \mathcal{M}$ [Bruns, Gubeladze, Henk, Martin and Weismantel 1999, Sebő 1990]. The solutions in $\mathcal{H}(M)$ are the *minimal* elements of $\mathcal{M} \setminus \{0\}$ with respect to the partial ordering \leq on \mathbb{N}^n defined by

$$(a_1, \dots, a_n) \leq (b_1, \dots, b_n) \text{ iff } a_i \leq b_i, \text{ for all } i = 1, \dots, n. \quad (3.36)$$

Alternatively, $\mathcal{H}(M)$ contains precisely the *non-decomposable* elements of \mathcal{M} , i.e. those elements in $\mathcal{M} \setminus \{0\}$ that cannot be written as the sum of two other elements of $\mathcal{M} \setminus \{0\}$. The solution set of a system of homogeneous linear diophantine *inequalities*

$$Ax \leq 0, x \in \mathbb{N}^n \quad (3.37)$$

defines again a submonoid of \mathbb{N}^n . This monoid is finitely generated by its non-decomposable elements, but not by its minimal elements. For example, the solution set of $x - y \leq 0, x, y \in \mathbb{N}$, is generated by the non-decomposable elements $(0, 1)$ and $(1, 1)$, but not by the minimal solution $(0, 1)$. Finally, any solution of the inhomogeneous system of linear diophantine *inequalities*

$$Ax \leq b, x \in \mathbb{N}^n \quad (3.38)$$

with $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$, can be obtained as the sum of a non-decomposable solution of the system $Ax - bz \leq 0, (x, z) \in \mathbb{N}^{n+1}$ with $z = 1$, and a non-negative integer linear combination of non-decomposable solutions of the homogeneous system (3.37) [Ajili 1998].

3.6.2. Algorithms.

Elliott [1903] was probably the first to develop a method for computing all minimal solutions $x \in \mathbb{N}^n$ of a linear diophantine equation

$$a_1x_1 + \dots + a_nx_n = 0, \quad a_1, \dots, a_n \in \mathbb{Z}. \quad (3.39)$$

He considered the *generating function*

$$\prod_{i=1}^n (1 + y_i \lambda^{a_i} + y_i^2 \lambda^{2a_i} + \dots) = \prod_{i=1}^n \frac{1}{1 - y_i \lambda^{a_i}}, \quad (3.40)$$

where y_1, \dots, y_n, λ are formal indeterminates. Expanding this function yields the sum of all terms of the form $y^u \lambda^\beta$, where $u \in \mathbb{N}^n$, $\beta \in \mathbb{Z}$ and $a_1u_1 + \dots + a_nu_n = \beta$. It follows that $u \in \mathbb{N}^n$ is a solution of (3.39) iff the expansion contains the term $y^u \lambda^0 = y^u$. The idea of Elliott is to rewrite this generating function until factors appear that do not contain λ . MacMahon [1916] extended this algorithm to the case of systems of homogeneous diophantine equations. Domenjoud and Tomás [1995] reformulated Elliott-MacMahon's approach as a transformation process on constrained parametric expressions and derived an algorithm for solving arbitrary linear diophantine constraints involving equations, inequalities, and disequalities. The algorithm computes a decomposition of the solution set into possibly overlapping subsets, obtaining a parametric representation of each subset from which minimal solutions can be extracted immediately. Another algorithm that computes a parametric representation was proposed by Abdulrab and Maksimenko [1995] based on a method of Makanin [1979], which applied only to systems of equations. In contrast to Domenjoud and Tomás' algorithm, the non-decomposable solutions may not occur explicitly in the parametric representation found. But, the subsets in the decomposition are pairwise disjoint, such that each solution is given in a unique way. A variant of the Elliott-MacMahon algorithm to compute the Hilbert Basis of a system of homogeneous linear diophantine equations is also presented in Pasechnik [1998].

Stanley [1973, 1986] found an explicit formula for the generating function and described a method to compute it. He analyzed the geometric structure of the polyhedral cone C defined by the non-negative real solutions of the system $Ax = 0$. The cone C can be represented as a finite union of simplicial cones. Domenjoud [1991] proposed an algorithm that looks for solutions in all simplicial subcones whose extreme rays are extreme rays of the solution cone C . As noted in [Tomás and Filgueiras 1997], Stanley's results imply that it is enough to consider a triangulation of the solution cone, which results in a much smaller number of subcones. The geometric structure of the solution cone was also used by [Tomás and Filgueiras 1997, Tomás 1997] to extend their *slopes algorithm*, which was originally developed for solving a single linear diophantine equation, to systems of equations. Another algorithm that they proposed is the *rectangles algorithm* [Filgueiras and Tomás 1993].

A second family of algorithms has been obtained by representing the partially ordered set (\mathbb{N}^n, \leq) as a directed acyclic graph and by searching for minimal so-

lutions in that graph. These methods are quite efficient when the elements of the Hilbert basis have a small norm. A first algorithm of this type was proposed by Huet and Lang [1978] for solving the homogeneous linear diophantine equation

$$\sum_{i=1}^n a_i x_i = \sum_{j=1}^m b_j y_j, \quad x_i, y_j \in \mathbb{N}, \tag{3.41}$$

where $a_i, b_j \in \mathbb{N}$. The basic idea is to perform a lexicographic enumeration of tuples in \mathbb{N}^{n+m} using the following bound on the components of minimal solutions

$$x_i \leq \max_{j=1, \dots, m} b_j \quad \text{and} \quad y_j \leq \max_{i=1, \dots, n} a_i. \tag{3.42}$$

Lambert [1987] substantially improved Huet’s bound by showing that

$$\sum_{i=1}^n x_i \leq \max_{j=1, \dots, m} b_j \quad \text{and} \quad \sum_{j=1}^m y_j \leq \max_{i=1, \dots, n} a_i, \tag{3.43}$$

see also [Henk and Weismantel 2000] for a recent generalization.

Fortenbacher [1993] proposed to direct the search for a solution by visiting a successor node only if its residue is closer to zero than that of the current node. More precisely, let $v \in \mathbb{N}^{n+m}$ be some tuple found by the algorithm that is not greater than or equal to some solution that has already been obtained. Then

$$v + e_k \text{ is a successor of } v \text{ iff } \text{res}(v) \text{res}(e_k) < 0, \tag{3.44}$$

where the starting nodes e_1, \dots, e_{n+m} form the canonical basis of \mathbb{R}^{m+n} , and the residue is given by $\text{res}(v) = \sum_{i=1}^n a_i v_i - \sum_{j=1}^m b_j v_{n+j}$. Guckenbiehl and Herold [1985] extended this algorithm to solve a single non-homogeneous linear diophantine equation. An improved version of both algorithms was developed in [Clausen and Fortenbacher 1989].

Contejean and Devie [1994] generalized Fortenbacher’s algorithm to apply to a general system of homogeneous linear diophantine equations $Ax = 0, x \in \mathbb{N}^n$ by replacing (3.44) with

$$v + e_k \text{ is successor of } v \text{ iff } (Av)^T Ae_k < 0, \tag{3.45}$$

The geometric interpretation of this condition is that $A(v + e_k)$ should lie in the halfspace orthogonal to Av containing the origin, which generalizes (3.44) to the m -dimensional case, where m denotes the number of equations to be solved (Fig. 9). While the correctness of this condition can be checked easily, the termination proof is technically involved and based on topological arguments [Contejean and Devie 1994].

Ajili [1994] proposed an algorithm for finding a complete representation of the solution set of a linear homogeneous inequality based on Fortenbacher’s algorithm. Rather than using a slack variable to reduce the problem to that of solving a linear homogeneous equation, the algorithm solves the inequality directly. Based on

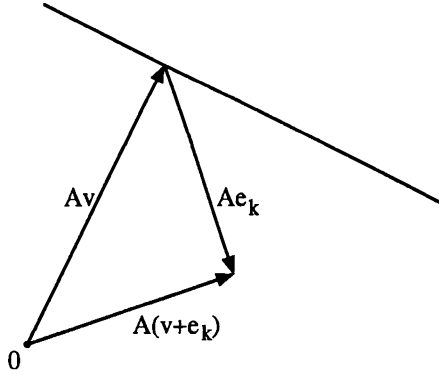


Figure 9: Criterion for determining successor nodes

this idea, Ajili and Contejean [1997] adapted the algorithm of Contejean and Devie [1994] to handle systems of linear equations and inequalities without introducing slack variables that would blow up the search space. This extension requires bounds on minimal solutions to ensure termination. [Ajili and Lock 1998] present an extended version of this algorithm that uses constraint propagation techniques to prune the search space.

The Gröbner basis approach for integer programming (see Section 3.5) can also be used to compute a Hilbert basis for a system $Au = 0, u \in \mathbb{N}^n$ [Pottier 1991a, Pottier 1991b, Sturmfels 1993]. Let G be the reduced Gröbner basis of the ideal

$$J_A = \langle x_1 - y_1 t_1^{a_{11}} \dots t_m^{a_{m1}}, \dots, x_n - y_n t_1^{a_{1n}} \dots t_m^{a_{mn}} \rangle \quad (3.46)$$

in the ring $\mathbb{Q}[x_1, \dots, x_n, y_1, \dots, y_n, t_1^{\pm 1}, \dots, t_m^{\pm 1}] = \mathbb{Q}[x_1, \dots, x_n, y_1, \dots, y_n, t_1, \dots, t_m, t_1^{-1}, \dots, t_m^{-1}] / (t_1 t_1^{-1} - 1, \dots, t_m t_m^{-1} - 1)$ with respect to a term order \succ such that $t_i \succ x_j \succ y_k$. Then the Hilbert basis of $Au = 0, u \in \mathbb{N}^n$ can be obtained as $G \cap \{x^u - y^u \mid u \in \mathbb{N}^n\}$.

Boudet and Comon [1996] show how to compute efficiently an automaton that accepts the set of solutions of a linear diophantine equation, and more generally of a system of linear diophantine equations, inequalities and disequalities. Applying Büchi's techniques for Presburger arithmetic, they obtain a decision procedure for Presburger arithmetic, which has nearly optimal worst case complexity, both for the existential fragment and for the full first-order theory (compare Section 3.8 for more precise worst-case bounds).

3.6.3. Bounds on the size and number of minimal solutions.

Many authors have deduced bounds on the size of minimal solutions of $Ax = 0, x \in \mathbb{N}^n$ in terms of the matrix A . Suppose $A \in \mathbb{Z}^{m \times n}$ has full row rank m and denote by Δ (resp. Δ_m) the maximum absolute value of a subdeterminant (resp. $m \times m$ subdeterminant) of A . Lambert [1987] gave for each minimal non-negative solution

of $Ax = 0$ the bound $\|x\|_\infty \leq n\Delta$. Pottier [1991a] and Domenjoud [1991] improved this independently to

$$\|x\|_\infty \leq (n - m)\Delta_m. \quad (3.47)$$

Other bounds in terms of the norm of A are, e.g. [Domenjoud 1991, Pottier 1991a]

$$\|x\|_\infty \leq (n - m) \left(\frac{\|A\|_1}{m} \right)^m \quad \text{and} \quad \|x\|_\infty \leq (n - m) m^{\frac{m}{2}} \|A\|_\infty^m, \quad (3.48)$$

where $\|A\|_1 = \sum_{i,j} |a_{ij}|$ and $\|A\|_\infty = \max_{i,j} |a_{ij}|$.

Henk and Weismantel [1997] derive an upper bound on the height of minimal Hilbert bases. Given a pointed polyhedral cone $C = \text{cone}(a^1, \dots, a^k)$, with $a^i \in \mathbb{Z}^n$, and an element b in the minimal Hilbert basis of C , the *height* of b is defined as

$$h_C(b) = \min \left\{ \sum_{i=1}^k \lambda_i \mid b = \sum_{i=1}^k \lambda_i a^i, \lambda_i \geq 0 \right\}. \quad (3.49)$$

Lankford [1989] gave an exponential lower bound for the number of minimal solutions of the linear diophantine equation $nx_{n+1} = x_1 + 2x_2 + \dots + nx_n$. Domenjoud [1992] derives an explicit formula for the number of minimal solutions of the equation

$$a \cdot \sum_{i=1}^n x_i = b \cdot \sum_{j=1}^m y_j, \quad a, b \in \mathbb{N}. \quad (3.50)$$

The complexity of recognizing and counting Hilbert bases is studied in [Durand, Hermann and Juban 1999] and [Hermann, Juban and Kolaitis 1999].

3.7. Linear 0-1 inequalities

Linear 0-1 inequalities

$$Ax \geq b, x \in \{0, 1\}^n \quad (3.51)$$

with $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}$, are special linear diophantine constraints, which are particularly interesting from a logical point of view. There are many remarkable parallels between theorem proving and mathematical programming. The logical concepts of resolution, extended resolution, input and unit refutation, the Davis-Putnam-Procedure, and drawing of inferences pertinent to a given topic are closely related to the mathematical concepts of cutting planes, Chvátal's method, elementary closure, branch and bound, and projection of a polytope, respectively [Hooker 1988b, Chandru and Hooker 1999, Hooker 2000].

For example, it is easy to see that any resolvent of a set of propositional clauses can also be obtained as a Gomory-Chvátal cutting plane from the corresponding clausal inequalities and the bound inequalities $0 \leq x_i, y_j \leq 1$. But, Gomory-Chvátal cutting planes are strictly more powerful. For example, the unsatisfiability of the famous *pigeon-hole problem* (fit n pigeons into $n - 1$ holes) can be proved by cutting planes in $O(n^3)$ steps [Cook, Coullard and Turán 1987], whereas any resolution

proof requires exponentially many steps [Haken 1985]. A long-standing open problem was to prove a superpolynomial lower bound for the cutting plane proof system. This was finally solved by [Pudlák 1997], see also [Haken and Cook 1999], who found classes of propositional tautologies that admit no polynomial cutting plane proof. There exist several other proof systems for linear 0-1 inequalities whose complexity has been investigated, see [Pudlák 1999].

Hooker [1988a] and [1992] generalized resolution from propositional clauses to arbitrary linear 0-1 inequalities. We sketch his method for the case of extended clauses. An *extended clause* is a linear 0-1 inequality of the form

$$L_1 + \cdots + L_m \geq d, \quad (3.52)$$

where $d \geq 1$ is a positive integer number and $L_i, i = 1, \dots, m$, is either a *positive literal* x_i or a *negative literal* $\bar{x}_i = 1 - x_i$. The intuitive meaning of (3.52) is that at least d out of the m literals L_i have to be true. Classical clauses or clausal inequalities correspond to the case $d = 1$. If $x_i, i = 1, \dots, l$, are the positive literals and $\bar{y}_j, j = 1, \dots, k$, are the negative literals, then the extended clause (3.52) can also be written in the form

$$x_1 + \cdots + x_l - y_1 - \cdots - y_k \geq d - k. \quad (3.53)$$

In many cases, extended clauses give a more compact representation of a set $S \subseteq \{0, 1\}^n$ than classical clauses. For example, the extended clause $L_1 + \cdots + L_m \geq d$ is equivalent to the conjunction $\bigwedge_{I \subseteq \{1, \dots, m\}: |I|=m-d+1} \sum_{i \in I} L_i \geq 1$ of $\binom{m}{m-d+1}$ classical clauses.

A classical clause $L_1 + \cdots + L_m \geq 1$ implies another clause $L'_1 + \cdots + L'_k \geq 1$ if and only if $L = \{L_1, \dots, L_m\} \subseteq L' = \{L'_1, \dots, L'_k\}$. A similar result holds for extended clauses. Abbreviate an extended clause $L_1 + \cdots + L_m \geq d$ by $L \geq d$ and view L as a set of literals $\{L_1, \dots, L_m\}$ of cardinality $|L| = m$. Then $L \geq d$ implies $L' \geq d'$ if and only if $|L \setminus L'| \leq d - d'$. This means that the entailment problem for extended clauses is easy, while for arbitrary linear 0-1 inequalities it is coNP-complete. *Generalized resolution* is described by the inference rules:

$$\text{Impl.Ext.Cl: } \frac{L \geq d}{L' \geq d'} \quad \text{if } |L \setminus L'| \leq d - d' \quad (3.54)$$

$$\text{Resolution: } \frac{L_i + L \geq 1, \bar{L}_i + L' \geq 1}{L + L' \geq 1}$$

$$\begin{array}{rcl} \square & L_2 & + \cdots + L_{m-1} + L_m \geq d \\ L_1 & \square & + \cdots + L_{m-1} + L_m \geq d \\ \text{Diagonal.Sum:} & \vdots & \\ & L_1 + L_2 + \cdots + L_{m-1} & \square \geq d \\ & \hline & L_1 + L_2 + \cdots + L_{m-1} + L_m \geq d + 1 \end{array} \quad (3.55)$$

Generalizing a classical result from Quine, Hooker [1992] showed that by applying these rules one can compute all prime extended clauses for a set of extended clauses C . An extended clause $L \geq d$ is *prime* for C if $L \geq d$ is implied by C and if there is no other extended clause implied by C that implies $L \geq d$. A set of extended clauses is unsatisfiable if and only one can derive by generalized resolution the empty clause $0 \geq 1$. Barth [1996] introduced *simplifying* resolvents and diagonal sums and showed how these can be computed efficiently. He also devised an algorithm to compute for an arbitrary linear 0-1 inequality an equivalent set of extended clauses. Bockmayr [1993, 1995] and Barth and Bockmayr [1995, 1998] study logic-based and polyhedral 0-1 constraint solving in the context of constraint logic programming.

3.8. Presburger arithmetic

Presburger arithmetic PA is the first-order theory of the integers in the language L having 0, 1 as constants, +, − as binary operations, and equality =, order \leq and congruences \equiv_n modulo all integers $n \geq 1$ as binary relations. So L -terms in PA can be written in the form of linear polynomials with integer coefficients. Atomic formulas are equations, inequalities or congruences between such L -terms. Formally, integer coefficients of variables and the constant 1 as well as moduli of congruences are written in unary notation. PA can be construed as the “linear fragment” of number theory. This relatively small fragment is nevertheless rich enough to express many important application problems, such as solvability of (parametric) systems of linear Diophantine equations, integer feasibility of systems of (parametric) linear constraints, integer programming, and certain problems in program description and verification [Gurari 1985, Suzuki and Jefferson 1980].

The fundamental paper of Presburger [1929] established an effective quantifier elimination procedure and decision procedure for PA. The strength of his results could be appreciated only by 1931, when Gödel’s incompleteness theorem showed that full first-order number theory (with multiplication), and hence even the “quadratic fragment” of first-order number theory is algorithmically undecidable [Gödel 1931, Gurari and Ibarra 1979]. As a by-product an analysis of the properties required in the course of a quantifier elimination procedure provides easily explicit axioms for PA: Presburger arithmetic can be axiomatized by first-order axioms saying that \mathbb{Z} is a discretely ordered Abelian group with smallest positive element 1 and that $|\mathbb{Z}/n\mathbb{Z}| = n$ for all natural numbers n . As a consequence, any lexicographical product $G = H \times \mathbb{Z}$ of a divisible ordered Abelian group H with \mathbb{Z} is a model of PA.

The complexity of the decision problem for fragments and extensions of PA has been studied extensively [Berman 1980, Cooper 1972, Ferrante and Rackoff 1979, Fischer and Rabin 1974, Fürer 1982, Gathen and Sieveking 1978, Grädel 1987, Gurari and Ibarra 1979, Gurari and Ibarra 1981, Lipshitz 1978, Oppen 1973, Reddy and Loveland 1978, Scarpellini 1984, Shostak 1979, Volger 1983]. Roughly speaking, the central results are the following: The validity of a prenex closed L -formula φ in the domain of integers can be decided in deterministic space bounded by $\ell^{n^{cb}}$,

for some constant c , where ℓ is the length of the input formula φ . Here n is the number of quantified variables in φ and b is the number of quantifier-blocks in the prefix of φ . More precisely, this decision can be made in time bounded by $\ell^{n^{cb}}$ by an alternating Turing machine with b alternations of quantifiers. A corresponding lower bound due to Fischer and Rabin [1974] asserts that there is a series $\{\varphi_n\}$ of closed formulas of length linear in n such that the validity of φ_n cannot be algorithmically decided in non-deterministic time 2^{2^n} . This lower bound is achieved by the powerful technique of coding the multiplication of a large initial segment of the positive integers by a short PA-formula. This enables one to transform the undecidability of the elementary theory of integers with addition and multiplication into a hardness result for PA.

An upper bound for the worst-case complexity of the quantifier elimination problem in PA was obtained in [Weispfenning 1990b]: Let ℓ be the length of the input formula φ , n the number of quantified variables in φ and b the number of quantifier-blocks in φ . Then the bound is of the form $\exp(\ell^{(cn)^b})$ for some constant c , where $\exp(x) = 2^x$. In [Weispfenning 1997a] it was shown that this upper bound is essentially tight; so the quantifier elimination problem for PA is inherently triply exponential. The lower bound uses the Fischer-Rabin technique [Fischer and Rabin 1974] together with the construction of a series of PA-formulas in one parameter that describe large finite sets in a short way.

An efficient quantifier elimination procedure for PA [Weispfenning 1990b] goes along the lines of elimination by test points sketched in Section 2.4.2. The fact that elements of \mathbb{Z} may not be divisible by a natural number n account for the fact that new congruences have to be introduced during quantifier elimination; moreover the number of test points required is much higher than for linear elimination in the reals.

This result seems to preclude any practical application of quantifier elimination in PA. However, the situation can be improved by exactly one exponential in exchange for a little sacrifice that is comparatively harmless in applications, namely an NP-search problem. The clue for this improvement is to abandon complete quantifier elimination in favour of bounded quantifier elimination, where one admits *bounded quantifiers* of the form $(\exists x, |x| \leq t), (\forall x, |x| \leq t)$, for some constant term t , in the output formulas.

This shift of viewpoint offers an extra advantage that is missing from complete quantifier elimination, namely *uniformity in the coefficients* of PA-terms and PA-formulas: Consider *Uniform Presburger arithmetic* UPA as the extension of PA, where we allow unquantified parameters in the coefficients of the integer variables. Formally this is achieved by introducing scalar terms of a new “scalar” sort that are built up from scalar parameters by certain scalar operations. Then the following results can be shown [Weispfenning 1997a]:

1. Complete quantifier elimination is impossible in UPA; in other words any quantifier elimination procedure for PA is necessarily non-uniform in the coefficients of the input. (This is in sharp contrast to a result of Dries and Holly [1992] that proves quantifier elimination for UPA *without order*.)

2. Bounded quantifier elimination for UPA has upper and lower worst-case complexity bounds that are exactly one exponential lower than those for quantifier elimination in PA.

Efficient bounded quantifier elimination for UPA can be used successfully for mixed integer optimization problems with linear constraints and linear or quadratic objective function, and problems of automatic loop parallelization in programs [Lengauer 1993].

Another extension of quantifier elimination for PA appears in [Weispfenning 1999]. It concerns mixed real-integer linear elimination. Let MRL be the language PA of Presburger arithmetic extended by the unary integer-part operation $\lfloor \cdot \rfloor$. Interpret this language over the real numbers with congruences being defined by $a \equiv_n b \iff b - a \in n\mathbb{Z}$. Then the elementary theory of reals in MRL admits effective quantifier elimination and hence is decidable. Upper complexity bounds for the quantifier elimination and decision problem are similar to those for Presburger arithmetic. So these results are a common generalization of corresponding results in Section 2.4.2 and 3.8. Further extensions to decision problems involving a transcendental function appear in [Anai and Weispfenning 2000, Weispfenning 2000].

4. Non-linear constraints over continuous domains

4.1. Zero-sets of univariate polynomials

In this section we describe exact symbolic algorithms that determine as closely as possible the structure of the set of common real or complex zeros of a finite set of univariate polynomials with real or complex coefficients, respectively. In particular these algorithms will determine the exact number of zeros and their location to any prescribed accuracy.

Consider a set F of univariate polynomials in $\mathbb{F}[x]$ and a univariate polynomial $G \in \mathbb{F}[x]$. Then $F = 0$ is the system of equations $\{f(x) = 0 \mid f \in F\}$. We denote the *greatest common divisor* of the polynomials in F by $\gcd(F)$. For $g \in \mathbb{F}[x]$ we let g^* denote the *square-free part* of g , i.e. the product of all different irreducible factors of g . We let $V_{\mathbb{K}}(F)$ denote the *variety* of F in some extension field \mathbb{K} of \mathbb{F} , i.e. the set of all common zeros of F in \mathbb{K} . For a single polynomial g we put $V_{\mathbb{K}}(g) = V_{\mathbb{K}}(\{g\})$. We say the equation $g = 0$ is *implied by* the system $F = 0$ over \mathbb{K} if $V_{\mathbb{K}}(F) \subseteq V_{\mathbb{K}}(g)$.

4.1. PROPOSITION (Solvability and entailment). *Let \mathbb{K} be an extension field of \mathbb{F} ; let $g = \gcd(F)$ and let g^* be the square-free part of g . Then the following hold:*

1. $V_{\mathbb{K}}(F) = V_{\mathbb{K}}(g) = V_{\mathbb{K}}(g^*)$. So the system $F = 0$ is solvable in \mathbb{K} iff g^* has a zero in \mathbb{K} . For an algebraically closed field \mathbb{K} , this is the case iff g^* is not in \mathbb{F} . Moreover the number of different solutions of this system equals the degree of g^* .
2. Let h be a further univariate polynomial in $\mathbb{F}[x]$. Then the equation $h = 0$ is implied by $F = 0$ if g^* divides h in the polynomial ring $\mathbb{F}[x]$. For an algebraically

closed field \mathbb{K} , the converse is also valid.

This is proved as follows: We let $\text{Id}(F)$ denote the *ideal* generated by F , i.e. the set of all linear combinations of polynomials in F using arbitrary polynomials from $\mathbb{F}[x]$ as factors. Then $\text{Id}(F) = \text{Id}(g)$, and so $V_{\mathbb{K}}(F) = V_{\mathbb{K}}(g)$. The equation $V_{\mathbb{K}}(g) = V_{\mathbb{K}}(g^*)$ follows directly from the definition of the square-free part. Since g^* splits into pairwise different linear factors over an algebraically closed field \mathbb{K} , the number of different zeros of g^* in \mathbb{K} equals the degree of g^* . If g^* divides h then clearly $V_{\mathbb{K}}(g^*) \subseteq V_{\mathbb{K}}(h)$. If \mathbb{K} is an algebraically closed field and $V_{\mathbb{K}}(g^*) \subseteq V_{\mathbb{K}}(h)$, then every linear factor of g^* in $\mathbb{K}[x]$ must also occur as linear factor in h . Since the linear factors of g^* are pairwise different, this means that g^* divides h .

Notice that for an algebraically closed field \mathbb{K} these criteria reduce the geometric questions whether $V_{\mathbb{K}}(F) = \emptyset$ or whether $V_{\mathbb{K}}(F) \subseteq V_{\mathbb{K}}(g)$ to purely arithmetic questions in the ring $\mathbb{F}[x]$. In order to solve these arithmetic questions algorithmically we use the *Euclidean algorithm for univariate polynomials* as a tool to compute $g = \text{gcd}(F)$ and g^* : The basic step of this algorithm is the reduction of a polynomial f by a polynomial g with $\deg(f) \geq \deg(g) \geq 0$: If f has highest monomial ax^m and g has highest monomial bx^n , then

$$f \xrightarrow{g} f_1 = f - a/bx^{m-n}g \quad (4.1)$$

and $\deg(f_1) < \deg(f)$ or $f_1 = 0$. This simple reduction step can be applied iteratively to a finite non-empty set F of non-zero polynomials in $\mathbb{F}[x]$ as follows: Choose $g \in F$ of minimal degree; replace all other polynomials $f \in F$ by their reducts f_1 w.r.t. g and discard all zero polynomials from the result. Iterate this procedure, until F contains only one polynomial g . Notice that in each iteration the maximal degree of all polynomials in F will decrease; so termination is guaranteed. Moreover, after each iteration the ideal generated by F , $\text{Id}(F)$, remains the same. So at the end we have $\text{Id}(F) = \text{Id}(g)$, which means that g is the greatest common divisor of all polynomials in F .

Next we can use another gcd computation, viz. $g^* = g / \text{gcd}(g, g')$, where g' is the derivative of g , to get the *square-free part* g^* of g . This computation of g^* avoids a complete factorization of g . Its correctness follows from the fact that the multiplicity of an irreducible factor in g reduces by one in $\text{gcd}(g, g')$. Notice that the proposition yields no information on the position of the zeros of g^* in $\mathbb{C} = \mathbb{R}^2$, since this would require inequalities between absolute values; in other words this is in fact a question about real zeros of two bivariate real polynomials representing the real and imaginary part of $g^*(x+iy)$ (comp. Section 4.2). Subsequent factorization of g^* into irreducible polynomials in $\mathbb{F}[x]$ [Mignotte 1992] may split $V_{\mathbb{C}}(g^*)$ into disjoint parts and hence may help actually to solve the equation by numeric methods.

In order to determine the number and position of the real zeros of g^* and hence of F one may proceed as follows: For two given non-zero polynomials $f, g \in \mathbb{R}[x]$ the iterated reduction of f by means of g as defined above will lead to the *remainder* $\text{rem}(f, g)$ of f upon division by g . Using iterated formation of *negative* remainders, one obtains the *Sturm-Sylvester sequence* $S(f, g)$ of (f, g) as follows: Define f_k recursively by $f_0 := f$, $f_1 := g$; $f_{k+1} := -\text{rem}(f_k, f_k)$. Then $S(f, g) = (f_0, f_1, \dots, f_k)$,

where f_k is the last non-zero polynomial in the recursive sequence. For a real number c let $\mathbf{f}(c)$ denote the sequence of real values taken by the polynomials in the sequence \mathbf{f} at the point c . Let $\text{sc}(\mathbf{f}(a))$ denote the number of sign changes in this sequence of real numbers, when zeros are ignored. Then a beautiful theorem due to Sturm and Sylvester [Ben-Or, Kozen and Reif 1986] says:

4.2. THEOREM. *Let f, g be non-zero relatively prime polynomials in $\mathbb{R}[x]$, let f be square-free, and let $a < b$ be real numbers that are non-zeros of f . Let \mathbf{f} be the Sturm-Sylvester sequence of $(f, f' \cdot g)$. Denote the number of zeros of f in the interval $[a, b]$, where g is either positive or negative, by $N(f, g, 1, a, b)$ and $N(f, g, -1, a, b)$ respectively. Then*

$$\text{sc}(\mathbf{f}(a)) - \text{sc}(\mathbf{f}(b)) = N(f, g, 1, a, b) - N(f, g, -1, a, b). \quad (4.2)$$

In particular, for $g = 1$

$$\text{sc}(\mathbf{f}(a)) - \text{sc}(\mathbf{f}(b)) \quad (4.3)$$

is the number of zeros of f in the interval $[a, b]$.

With the obvious sign evaluation of real polynomials at $\pm\infty$ this theorem can also be applied to the case $a = -\infty$ and/or $b = \infty$. The proof employs only the most elementary algebraic facts about real polynomials. By applying the theorem to the pairs f, g and $f, 1$ one easily obtains the separate numbers $N(f, g, 1, a, b), N(f, g, -1, a, b)$. The hypothesis that f is square-free and g is relatively prime can be circumvented by first computing the square-free part f^* of f and $h = f^* / \gcd(f^*, g)$, and then applying the theorem to h in place of f .

Another calculation of $N(f, g, 1, a, b) - N(f, g, -1, a, b)$ due to Hermite has recently found a beautiful generalization to the multivariate case (comp. [Becker and Wörmann 1994, Pedersen, Roy and Szpirglas 1993] and Section 4.2 below): It does not require f to be square-free and g to be relatively prime to f . It asserts that the number $N(f, g, 1, a, b) - N(f, g, -1, a, b)$ is the signature (i.e. the number of positive eigenvalues minus the number of negative eigenvalues) of a symmetric $n \times n$ matrix (a_{ij}) that is computed as follows: $n = \deg(f)$ and a_{ij} is the trace of the linear map $h \mapsto x^{i+j}gh$ on the real vector space $\mathbb{R}[x]/f$ of residues modulo f with respect to the canonical basis $\{1, x, \dots, x^{n-1}\}$.

Moreover, there is a far-reaching generalization of the corollary to the counting of real zeros c of f satisfying several side conditions, say $g_1(c) > 0, \dots, g_m(c) > 0$. Again this generalization is derived in a purely combinatorial way by applying the Sturm-Sylvester theorem to finitely many pairs of polynomials (f, \bar{g}) , where \bar{g} is a product of certain g_i and its squares. The number of these pairs is a priori exponential in the number m of side conditions. This exponential growth can be avoided by a combinatorial divide-and-conquer argument [Ben-Or et al. 1986]. Since over the reals the common zeros of a finite polynomial system $F = \{f_1, \dots, f_r\}$ can be expressed as the zeros of a single polynomial $f = f_1^2 + \dots + f_r^2$, this generalization solves all point counting problems for finite systems of univariate polynomial inequalities.

A priori bounds for the size of the real zeros of a univariate polynomial in terms of its coefficients are given by the *Cauchy bounds* [Mishra 1993]. Other a priori bounds are known e.g. for the distance between roots [Mignotte 1992].

Once a Sturm-Sylvester sequence for (f, f') has been computed for a square-free polynomial f , one can evaluate the number of sign changes of $S(f, f')$ at the endpoints of intervals obtained from the starting interval $[-C, C]$, where C is a rational number greater than a Cauchy bound for f by successive bisection until one has found a finite sequence of pairwise disjoint open intervals with rational endpoints each of which contains exactly one real root of f . Call these the *isolating intervals* for the real roots of f . Using only the endpoints a, b of such an isolating interval and the coefficients of f , one can now evaluate any other polynomial $g \in \mathbb{Q}[x]$ at the unique zero α of f in the open interval $]a, b[$ using again the theorem of Sturm-Sylvester: Assume without restriction that f is square-free and let \mathbf{h} be defined as above. Then

$$g(\alpha) > 0 \text{ (} = 0, < 0 \text{)} \iff \text{sc}(\mathbf{h}(a)) - \text{sc}(\mathbf{h}(b)) = 1 \text{ (} = 0, = -1 \text{)}. \quad (4.4)$$

This shows that the *isolating interval* $]a, b[$ for α can be used as an exact algebraic code for the algebraic real number α in algebraic computations. Moreover by iterating the bisection of intervals and the evaluation of the number of sign changes of $S(f, f')$ at the endpoints of these intervals, one obtains arbitrarily good numeric approximations of α . This coding of real algebraic numbers is used extensively in the SAC-2 computer algebra system [Collins 1985]. Another possible unique code for a real zero α of f is the sequence of signs of all higher derivatives of f at α (Thom's Lemma [Roy 1996]). The latter has the advantage that it works in arbitrary, not necessarily Archimedean, real-closed fields.

4.2. Zero-sets of multivariate polynomials

In the univariate case of Section 4.1, the computation of the variety of a finite system F of polynomials was reduced to that of a single polynomial by taking a greatest common divisor. While gcd's still can be computed in multivariate polynomial rings R over fields, they do not serve the same purpose for solving equations, since $\text{gcd}(F)$ is no longer a R -linear combination of polynomials in F , because R is not a principal ideal ring. So the role of the greatest common divisor is taken over by the ideal generated by F . The fundamental transition from geometric properties of the variety of F in some extension field \mathbb{K} of \mathbb{F} to arithmetic properties of polynomials in the polynomial ring R over the base field \mathbb{F} is achieved by a "*Nullstellensatz*", i.e. a theorem on zeros for the field \mathbb{K} .

We consider the ring $R = \mathbb{F}[x_1, \dots, x_n]$ of multivariate polynomials in the indeterminates x_1, \dots, x_n over \mathbb{F} . As in Section 4.1, we let $\text{Id}(F)$ denote the *ideal* generated by F , i.e. the set of all linear combinations of polynomials in F using arbitrary polynomials from R as factors. We let $V_{\mathbb{K}}(F)$ denote the *variety* of F in some extension field \mathbb{K} of \mathbb{F} , i.e. the set of all common zeros of F in \mathbb{K}^n . For a single polynomial g we put $V_{\mathbb{K}}(g) = V_{\mathbb{K}}(\{g\})$. We say the equation $g = 0$ is *implied*

by the system $F = 0$ over \mathbb{K} if $V_{\mathbb{K}}(F) \subseteq V_{\mathbb{K}}(g)$. To begin with we state *Hilbert's Nullstellensatz* for algebraically closed fields. It applies in particular to the case $\mathbb{F} = \mathbb{K} = \mathbb{C}$.

4.3. THEOREM (Solvability and entailment in algebraically closed fields). *Let \mathbb{K} be an algebraically closed extension field of \mathbb{F} , let $R = \mathbb{F}[x_1, \dots, x_n]$, and let $I = \text{Id}(F)$ denote the ideal generated by F in R . Then the following hold:*

1. $V_{\mathbb{K}}(F) = V_{\mathbb{K}}(I)$.
2. The system $F = 0$ is solvable in \mathbb{K}^n iff $1 \notin I$.
3. Let g be a further polynomial in R . Then the equation $g = 0$ is implied by $F = 0$ iff there exists a $k \in \mathbb{N}$ such that $g^k \in I$.
4. Let g be a further polynomial in R , let y be a new indeterminate, and let $S = \mathbb{F}[x_1, \dots, x_n, y]$. Then the equation $g = 0$ is implied by $F = 0$ iff $1 \in \text{Id}(I \cup \{1 - yg\})$, where the ideal is formed in S .

The first statement is obvious. The second is an immediate consequence of the third by taking $g = 1$. The last statement is an immediate consequence of the second; it is known as the *Rabinovich trick*. In the third statement the implication from right to left is again obvious. The converse is a theorem of considerable depth that requires either a considerable amount of ideal theory [Becker et al. 1993] or an elementary but intricate induction on the number of indeterminates [Seidenberg 1956b, Seidenberg 1956a]. The second and the last statement reduce the relevant geometrical questions to the *ideal membership problem* in polynomial rings over \mathbb{F} . An algorithmic solution of this problem is offered by the construction of Gröbner bases described below in Section 4.2.2.

In order to classify the varieties $V_{\mathbb{K}}(I)$ for an algebraically closed field \mathbb{K} , it is natural to consider the (maximal) geometric dimension occurring in $V_{\mathbb{K}}(I)$. This concept is captured algebraically by the *dimension* of the ideal I : Let $\mathcal{Y} \subseteq \{x_1, \dots, x_n\}$ be a set of indeterminates; then \mathcal{Y} is called *independent* modulo I if $\mathbb{F}[\mathcal{Y}] \cap I = \{0\}$. The dimension of the ideal I is then the cardinality of the largest independent set of indeterminates. Again Gröbner basis theory will provide an algorithmic way of computing the dimension of an ideal (see Section 4.2.2 below).

Hilbert's Nullstellensatz has the following counterpart for real closed fields, which is valid in particular for the field \mathbb{K} of reals. Recall that an ordered field \mathbb{K} is real closed if the intermediate value theorem for polynomials holds in \mathbb{K} .

4.4. THEOREM (Solvability and entailment in real closed fields). *Let \mathbb{K} be a real closed extension field of \mathbb{F} , let $R = \mathbb{F}[x_1, \dots, x_n]$, and let $I = \text{Id}(F)$ denote the ideal generated by F in R . Then the following hold:*

1. $V_{\mathbb{K}}(F) = V_{\mathbb{K}}(I)$.
2. The system $F = 0$ is solvable in \mathbb{K}^n iff for all $m \in \mathbb{N}$, all positive $p_1, \dots, p_m \in \mathbb{F}$, and all $h_1, \dots, h_m \in R$, $1 + \sum_{i=1}^m p_i h_i^2 \notin I$.
3. Let g be a further polynomial in R . Then the equation $g = 0$ is implied by $F = 0$ iff there exist $k, m \in \mathbb{N}$, positive $p_1, \dots, p_m \in \mathbb{F}$, and $h_1, \dots, h_m \in R$, such that $g^{2k} + \sum_{i=1}^m p_i h_i^2 \in I$.

Here again the first statement, the implication from left to right in statement two, and the implication from right to left in statement three are obvious. Moreover, statement two is the special case of statement three, where $g = 1$. The implication from left to right in statement three is a deep fact of real algebra [Bochnak, Coste and Roy 1987, Weispfenning 1975]. While the real Nullstellensatz is formally analogous to Hilbert's Nullstellensatz, it does not adapt as easily to algorithmic procedures due to the presence of the p_i , and h_i [Lombardi 1991].

Next we study algorithmic approaches to the questions addressed in the theorems. What is required is a replacement of the $\gcd(F)$ in the solution of univariate systems of equations. In fact, the role of the greatest common divisor can be simulated to some extent by certain finite polynomial systems G that have the same R -linear combinations as F , but exhibit the structural properties of the complex or real variety of F more clearly.

There are two essentially different generalizations of the Euclidean algorithm (and hence the \gcd -concept):

- The computation of *characteristic sets* (Ritt-Wu) [Chou 1988, Mishra 1993] and
- the computation of *Gröbner bases* (Buchberger) [Becker et al. 1993, Cox, Little and O'Shea 1992, Cox, Little and O'Shea 1998].

They are based on two different generalizations of the univariate reduction step to the multivariate case corresponding to the recursive and the distributive representation of multivariate polynomials. In the recursive representation, polynomials f over a field \mathbb{F} in indeterminates x_1, \dots, x_n are viewed as objects in $(\dots(\mathbb{F}[x_1])\dots)[x_n]$, whereas in the distributive representation, f is regarded as a \mathbb{F} -linear combination of *terms*, i.e. power-products $x_1^{e_1} \dots x_n^{e_n}$ of the indeterminates. We begin with a sketch of the *characteristic set method*.

4.2.1. Characteristic Sets

In the recursive representation of polynomials one can define a quasi-order on polynomials by putting $f < g$ if g contains an indeterminate with higher index than the highest indeterminate occurring in f , or both have the same highest indeterminate x_i , but g has higher degree in x_i than f . If f is written as univariate polynomial in its highest indeterminate x_j with coefficients in the remaining indeterminates, then the highest coefficient of f is called the *initial* I_f of f . Suppose now g is another polynomial with $m = \deg_{x_j}(g) \geq \deg_{x_j}(f) = k$ and g is written as univariate polynomial in the indeterminate x_j with coefficients in the remaining indeterminates. (We do not require x_j to be the highest indeterminate in g .) Then we have the following (*Ritt*-)reduction of g by the *divisor* f :

$$g \xrightarrow{f} g^* = I_f g - I_g x_j^{m-k+1} f \quad \text{with} \quad \deg_{x_j}(g) \geq \deg_{x_j}(g^*). \quad (4.5)$$

g is called *reduced w.r.t. f* if no such reduction is possible, i.e. if $\deg_{x_j}(g) < \deg_{x_j}(f)$. Notice that any chain of reductions using the same divisor f will terminate after finitely many steps, since \deg_{x_j} decreases in each step.

An *ascending set* is either a non-zero constant or a finite sequence A of non-constant polynomials with strictly increasing highest indeterminates, such that each polynomial in A is reduced w.r.t. all the previous ones. Ascending sets are quasi-ordered by first differences in their entries from left to right. If no such difference in quasi-order is encountered, the longer sequence is declared smaller than the shorter one. This yields a well-founded quasi-order on ascending sets. As a consequence, every non-empty set F of polynomials in the given polynomial ring contains a minimal ascending set C . Any such C is called a *characteristic set* of F . For finite F such a characteristic set can be determined in an obvious way by inspection.

Let now F be finite and let $C \subseteq F$ be a characteristic set of F . Then each polynomial $f \in F$ can be reduced as long as possible, using the polynomials in C successively in decreasing order, to a polynomial f^* (which may coincide with f if no such reduction is possible). Call f^* the *remainder* of f upon reduction by the ascending set C . Adding all the non-zero remainders f^* of polynomials $f \in F$ obtained in this way to the set F , we obtain a finite set F_1 extending F . If F_1 is strictly larger than F , then F_1 contains a new characteristic set C_1 smaller than C in the quasi-order of ascending sets.

Iterating this procedure we arrive at an increasing chain of polynomial sets $F = F_0 \subset F_1 \subset F_2 \subset \dots$ with an associated strictly decreasing chain of characteristic sets $C = C_0 > C_1 > C_2 > \dots$. Since the quasi-order on characteristic sets is well-founded, this iteration will terminate after finitely many steps, say with F_k and C_k . By construction, every polynomial in F_k is still in the ideal $\text{Id}(F)$, and so $\text{Id}(F_k) = \text{Id}(F)$, and so $V_{\mathbb{K}}(F_k) = V_{\mathbb{K}}(F)$ for $\mathbb{K} = \mathbb{C}$ or $\mathbb{K} = \mathbb{R}$. Moreover, the remainder of every $f \in F_k$ upon reduction by C_k is zero. As a consequence, we have the following relation between the variety of F_k and the variety of C_k : Let J be the product of all the initials of the polynomials in C_k . Then

$$V_{\mathbb{K}}(F) \setminus V_{\mathbb{K}}(J) = V_{\mathbb{K}}(C_k) \setminus V_{\mathbb{K}}(J) \quad \text{for } \mathbb{K} = \mathbb{C} \text{ or } \mathbb{K} = \mathbb{R}. \quad (4.6)$$

Call the characteristic set $C' = C_k$ of $F' = F_k$ obtained in this way an *extended characteristic set* of F . Notice that C' has triangular shape. So $V_{\mathbb{C}}(C')$ is finite iff every x_j is the highest indeterminate of the j -th polynomial in C' . If this is the case, then the individual points in $V_{\mathbb{C}}(C')$ can be computed e.g. numerically by solving successive univariate polynomials. In particular $V_{\mathbb{C}}(C') = \emptyset$ iff C' consists of a constant only. If C' contains less than n polynomials, there will be gaps in the sequence of highest variables of these polynomials, which are related to the dimension of $V_{\mathbb{C}}(C')$. So we have got some structural information on the set $V_{\mathbb{C}}(F) \setminus V_{\mathbb{C}}(J) = V_{\mathbb{C}}(C_k) \setminus V_{\mathbb{C}}(J)$.

In order to determine the remaining part of $V_{\mathbb{C}}(F)$, i.e. $V_{\mathbb{C}}(F) \cap V_{\mathbb{C}}(J)$, we proceed as follows: For each initial I_h of some polynomial h in C' , we consider the set $F_h = F' \cup \{I_h\}$ and compute as before an extended characteristic set C''_h . By construction $C''_h < C'$.

Iterating these steps of adding initials and computing extended characteristic sets will lead to a finitely branching tree of ascending sets. Each branch of the tree is finite, since the ascending sets in this branch are decreasing, and so by König's lemma the whole tree is finite. Let C_1, \dots, C_m be the extended characteristic sets

that form the leaves of this tree. Then for any extension field \mathbb{K} of \mathbb{F} we have a disjoint decomposition

$$V_{\mathbb{K}}(F) = \bigcup_{i=1}^m (V_{\mathbb{K}}(C_i) \setminus V_{\mathbb{K}}(J_i)), \quad (4.7)$$

where the systems C_i are ascending sets (and hence of triangular form) and the J_i are products of all initials of the polynomials in C_i .

This decomposition can then be used for numeric computations of individual elements of $V_{\mathbb{C}}(F)$ and of $V_{\mathbb{R}}(F)$ in case these sets are finite. Alternatively, individual solutions in \mathbb{C} can be computed symbolically in iterated algebraic extension fields of \mathbb{Q} . This requires, however, factorization of polynomials over iterated algebraic extension fields and can get quite complex. Individual real solutions can be computed symbolically from the decomposition using their description by isolating rational intervals described in the previous section.

The method has been employed most successfully in order to show for given polynomial f that $V_{\mathbb{C}}(F) \subseteq V_{\mathbb{C}}(f)$ and hence $V_{\mathbb{R}}(F) \subseteq V_{\mathbb{R}}(f)$ by showing that f Ritt-reduces to zero w.r.t. all C_i . This yields automatic proofs of many interesting geometrical theorems [Chou 1988, Mishra 1993].

Next we give a quick sketch of the *Gröbner basis method* and its applications to equation solving [Becker et al. 1993].

4.2.2. Gröbner Bases

The basic idea of the Gröbner basis method is similar to the Ritt-Wu method: namely to modify a given finite system of polynomials F to another finite system G with the property that the varieties of F and G are closely related, and structural properties of the complex and real variety of G can be derived from the shape of G rather directly. In contrast to the construction of characteristic sets, the construction of a Gröbner basis G from F does not yield a decomposition of the variety of F . Instead, the variety remains the same, $V_{\mathbb{K}}(G) = V_{\mathbb{K}}(F)$. In fact even the ideal generated by F is preserved, $\text{Id}(F) = \text{Id}(G)$.

A central concept is again some kind of polynomial reduction, here *Buchberger reduction*. Recall that Ritt-reduction regarded polynomials as univariate polynomials in the highest indeterminate allowing multiplication of the dividend with polynomials in lower variables. For Buchberger reduction we think of polynomials as finite \mathbb{Q} -linear combinations of terms (power-products of the x_i). In order to perform some kind of division with remainder, we should know what we shall regard as the “highest” term in the divisor polynomial.

This is achieved by introducing a *term order*, i.e. a linear order on the set T of terms which has 1 as smallest element and for which multiplication is monotonic. Examples are the *lexicographic* term order (induced by the lexicographic order of the exponent tuples), the *inverse lex* term order (just as the lex order with the order of variables reversed), the *degree-lex* term order (which orders by total-degree first and breaks ties using the lex order on the set T of terms), and the *degree-reverse-lex*

term order (which orders by total-degree first and breaks ties using the lex order backward on the set T of terms). Given a fixed term order, we can speak of the “highest monomial” $\text{HM}(f)$ of a non-zero polynomial f just as in the univariate case. (By convention all monomials will have non-zero coefficients.)

The resulting reduction concept is straightforward:

$$g \xrightarrow[f]{} g_1 = g - \frac{b}{a}uf, \quad (4.8)$$

if g contains a monomial bt ($b \in \mathbb{K}, t \in T$) and $\text{HM}(f) = as$ and $t = us$ in T . Iterated Buchberger reduction of a polynomial g using divisors f from a given set F is denoted by $g \xrightarrow[F]{*} h$. Any chain of reductions is finite. This can be seen as follows: By Dickson’s lemma, divisibility on T is a well-quasi-order; this is just another way of saying that every non-empty subset of \mathbb{N}^n has a finite Hilbert basis (compare Section 3.6). Thus Buchberger reduction is a terminating reduction relation on $\mathbb{K}[x_1, \dots, x_n]$ in the sense of term rewriting. Since any term order refines the divisibility relation on T , term orders are well-orders; any reduction decreases the highest term of the polynomials w.r.t. the chosen term order.

At first glance, Buchberger reduction for the inverse lexicographical term order appears to be very close to Ritt reduction. Notice that Ritt reduction will be applicable much more often than Buchberger reduction since it allows multiplication of the dividend by the initial of the divisor. So it may not be surprising that the addition of all remainders of polynomials in F by iterated Buchberger reduction with divisors in F will in general (except in the univariate case) not yield much more structural information on $\text{Id}(F)$ and hence on $V_{\mathbb{K}}(F)$.

So another process is needed for generating new polynomials in $\text{Id}(F)$. It takes account of cancellations of highest monomials in R -linear combinations of elements of F that are not discovered by the reduction process. Take e.g. the set $F = \{f, g\} \subset \mathbb{Q}[x, y]$ with $f = x^2y + 1$, $g = xy^2 + 1$. Here no Buchberger reduction is possible, but $yf - xg = y - x \in \text{Id}(F)$. For every pair of polynomials $f, g \in \mathbb{F}[x_1, \dots, x_n]$ there is an obvious way of constructing the “minimal” such cancellation, as in the example. Call the resulting “subtraction”-polynomial the *S-polynomial* $\text{spol}(f, g)$ of f and g .

The construction of a Gröbner basis G (w.r.t a fixed term order) from F proceeds by an iteration of adding all non-zero remainders of all S-polynomials w.r.t. reduction using the set of polynomials at hand as possible divisors. Again Dickson’s lemma guarantees that this iteration stops after finitely many steps. The output will be a finite set G with $\text{Id}(G) = \text{Id}(F)$ and the property that all S-polynomials of pairs from G can be reduced to zero using polynomials from G . Such a set G is called a *Gröbner basis* of $\text{Id}(F)$ (w.r.t the given term order). There are many equivalent definitions of Gröbner bases [Becker et al. 1993]; one of them asserts that G is a Gröbner basis iff Buchberger reduction w.r.t. G is confluent in the sense of term rewriting.

In practice the computation of normal forms of S-polynomials w.r.t. the Buchberger reduction is the bottleneck of the construction, since it may involve computations with very large coefficients. So a number of sophisticated criteria and methods

has been developed in order to recognize without lengthy computations superfluous S -polynomials, i.e. S -polynomials whose normal form will not contribute to the construction of the Gröbner basis.

The general strategy of computing Gröbner bases by adding normal forms of S -polynomials may be regarded as a critical-pair (or Knuth-Bendix) completion procedure as defined in term rewriting. Notice, however, that polynomials in the Buchberger algorithm correspond to ground terms, i.e. variable-free terms in term rewriting. For a detailed study of Gröbner bases in the framework of term rewriting see [Le Chenadec 1986, Bündgen 1996, Madlener and Reinert 1998].

In order to simplify a given Gröbner basis G one may interreduce all polynomials in G (by Buchberger reduction) as long as possible. This process terminates as well and yields a *reduced Gröbner basis* of $\text{Id}(F)$. Rather surprisingly reduced Gröbner bases are uniquely determined by $\text{Id}(F)$ and the term order, provided all polynomials are made monic.

What is the information on $V_C(F)$ that can be obtained from a Gröbner basis G of $\text{Id}(F)$ without much further effort?

- A solution of the ideal membership problem: $g \in \text{Id}(F) \iff g$ reduces to zero w.r.t. G .
- A solution of the word problem for the congruence relation generated by $I = \text{Id}(F)$: $g \equiv_I h \iff g - h$ reduces to zero w.r.t. G .
- The decision, whether $V_C(F) = \emptyset$ (check if G contains a non-zero constant).
- The decision, whether $V_C(F)$ is finite (check if for all $1 \leq i \leq n$ some power of x_i occurs in $\text{HM}(g)$ for some $g \in G$). If this is not the case, the computation of the dimension of $V_C(F)$ and of a maximal set of independent variables (inspection of the highest terms in G [Becker et al. 1993, Kredel and Weispfenning 1988]).
- If $V_C(F)$ is finite, explicit computations in the residue class ring $\mathbb{Q}[x_1, \dots, x_n]/\text{Id}(F)$, computation of the radical of $\text{Id}(F)$, computation of the number of complex zeros of F .

For a set F of homogeneous polynomials, $V_C(F)$ is always non-empty, since it contains the origin. Here the modified problem is to determine, whether the set $V_C^*(F)$ of non-trivial common complex zeros of F is non-empty. By the homogeneity of F this is the case iff $V_C(F)$ is infinite. So by the above this holds iff for some $1 \leq i \leq n$ no power of x_i occurs in $\text{HM}(g)$ for any $g \in G$.

All the above works for arbitrary term orders. Other applications such as the computation of elimination ideals [Becker et al. 1993] requires a block-order, where the indeterminates are grouped into two blocks such that the first block is lexicographically smaller than the second. The complexity of a Gröbner basis computation for a given system F may vary enormously according to the term order chosen. There are some heuristic principles but no definite theoretical results for the choice of a “good” term order. For numerical (or symbolic algebraic) computation of individual solutions, one often requires the reduced Gröbner basis G to be in triangular form; this is guaranteed for the lexicographical term orders. For a detailed study of solutions of algebraic systems in triangular form by symbolic methods, see [Lazard 1991, Lazard 1992].

Moreover, Gröbner bases have numerous other applications in computational commutative algebra [Becker et al. 1993]. At present computations of $V_{\mathbb{C}}(F)$ via Gröbner bases are limited – as a rule of thumb – to systems of say at most 10 variables and of total degree at most 10. This scope can sometimes be extended considerably by combining the Gröbner bases construction with intermediate factorization of polynomials [Melenk, Möller and Neun 1989, Gräbe 1994]. This yields a decomposition of $V_{\mathbb{C}}(F)$ as a finite union of smaller varieties $V_{\mathbb{C}}(G_i)$ given by their respective Gröbner bases G_i .

For zero-dimensional systems F (i.e. systems with finite complex variety) the computation of an *involution basis* provides sometimes a faster alternative to the Buchberger algorithm for the computation of a Gröbner basis [Zharkov and Blinkov 1993, Apel 1995, Apel 1998]. The method is an adaptation of ideas developed by Janet for non-commutative rings of differential operators to the case of commutative polynomial rings.

In case $V_{\mathbb{C}}(F)$ is finite, the real zero set $V_{\mathbb{R}}(F)$ can be computed e.g. as follows, compare [Becker et al. 1993, Section 8.8]: One introduces a new variable Z as random linear combination ℓ of x_1, \dots, x_n such that the variety $V_{\mathbb{C}}(F \cup \{\ell\})$ is in normal position with respect to Z , i.e. different zeros of this system have different Z -components. This fact is checked by a Gröbner basis computation. Then Z satisfies a univariate polynomial equation $f(Z) = 0$ over \mathbb{Q} and all x_i are given as polynomial functions of Z . Hence the real zeros of F can now be determined from the real zeros of the univariate polynomial f .

An alternative method dealing directly with a Gröbner basis of a real zero dimensional ideal is based on a *multivariate version of the Sturm-Sylvester theorem* discovered recently [Becker and Wörmann 1994, Pedersen et al. 1993]. It uses the Gröbner basis to compute the signature of a quadratic form on the residue ring defined by I , similar to the univariate case described in the previous section.

4.3. Parametric zero-sets

In many applications of constraint solving one wants to study not only the zero sets of fixed polynomial systems, but also a whole class of such problems given by a polynomial system $F = 0$, where the coefficients of the polynomials in the system depend on (complex or real) parameters. Then the complex or real zero-set of F depends on these parameters and the goal is to describe this dependence on the parameters as uniformly as possible.

Another source of parametric zero-sets are polynomial systems F that are parameter-free but where $V_{\mathbb{C}}(F)$ has positive dimension and one wants to determine the zeros of F depending on a set $\mathcal{Y} \subseteq \{x_1 \dots x_n\}$ of independent variables acting as parameters. This situation may be subsumed under the former case by regarding the variables in \mathcal{Y} as parameters.

So we consider a finite set $F \subseteq \mathbb{Q}[U_1, \dots, U_m][x_1, \dots, x_n]$, and we want to determine $V_{\mathbb{C}}(F_{\mathbf{a}})$ and $V_{\mathbb{R}}(F_{\mathbf{a}})$ depending on arbitrary complex or real values $\mathbf{a} = (a_1, \dots, a_m)$ of the parameters $\mathbf{U} = (U_1, \dots, U_m)$. Here $F_{\mathbf{a}} = \{f(\mathbf{a}, \mathbf{x}) \mid$

$f(\mathbf{U}, \mathbf{x}) \in F$ }. What is the answer we expect for questions like $|V_{\mathbb{C}}(F_{\mathbf{a}})| = d$ or $|V_{\mathbb{R}}(F_{\mathbf{a}})| = d$ or $\dim V_{\mathbb{C}}(F_{\mathbf{a}}) = d$ for a natural number d , for arbitrary values of \mathbf{a} ? Obviously it should consist of conditions on the parameters \mathbf{U} that are easy to evaluate for a given value \mathbf{a} of \mathbf{U} . Typically such a condition is a Boolean combination of polynomial equations in \mathbf{U} in the complex case, and of a Boolean combination of polynomial inequalities in the real case. Such a condition will then constitute a *uniform solution* to the parametric problem. It is well-known from algebraic model theory that conditions of this form do indeed exist. In the following, we sketch some methods that can be useful for computing such conditions in practice.

The basic idea is to use the methods applied in the non-parametric cases in such a way as to split the computation into a tree depending on case distinctions concerning the vanishing (or the signs in the real case) of certain coefficients that enter the computation step in an essential way. In principle it suffices to treat systems of parametric univariate polynomial equations and disequations (equations and inequalities in the real case), since the case of several main variables can then be handled by induction. Recall from the univariate case in Section 4.1 that the Euclidean algorithm and its variants (Sturm-Sylvester sequences) form the centerpiece of the non-parametric univariate algorithms. Unfortunately in the parametric case, the Euclidean algorithm will require an enormous number of case distinctions on the possible vanishing of highest coefficients of polynomials used as divisors. So in general it is not advisable for complexity reasons to follow this route, see however the next section for a counterexample. Various replacements for the Euclidean algorithm and Sturm-Sylvester sequences have been considered in the literature that behave better in the parametric case. Important concepts arising in this problem area are various types of *resultants*, *subresultants* and for real zeros *Sturm-Habicht sequences* [González, Lombardi, Recio and Roy 1989, González, Lombardi, Recio and Roy 1990, González-Vega, Recio, Lombardi and Roy 1998, González-Vega 1998, Renegar 1991, Kapur and Lakshman 1992, Cox et al. 1998].

For multivariate parametric systems, the method for computing an extended characteristic set is, by its inherent case distinctions, well fitted for parameters and does not require essential changes from the non-parametric case [Gao and Chou 1991, Kapur 1995a]. But again the number of case distinctions may be prohibitive in practice. For parametric systems of linear equations, a symbolic solution method has been presented in [Sit 1991].

For the rest of this section we consider the use of *Gröbner bases in the parametric case* and of various types of *resultants*.

4.3.1. Comprehensive Gröbner Bases

The basic observation here is that it is not sufficient to compute a Gröbner basis G of F with respect to the variables \mathbf{x} and indeterminate parameters \mathbf{U} , since under specialization of \mathbf{U} to some value \mathbf{a} , $G_{\mathbf{a}}$ may no longer be a Gröbner basis. This problem is remedied by the more subtle construction of a *comprehensive Gröbner basis* for F that does in fact remain stable under specialization of parameters [Weispfenning 1992]. The idea of the construction is to perform a tree of

Gröbner basis computations w.r.t. the necessary case distinctions on the parameters, but to keep all the coefficients in the computation that have been asserted to vanish in some case. In this manner all the Gröbner bases obtained at the leaves of the tree stay inside the given input ideal; so the whole case distinction can be discarded and the different Gröbner bases at the leaves can be joint into one single polynomial system which turns out to be a comprehensive Gröbner basis of F . Alternatively, one may attach to each node in the tree the corresponding condition on the parameters that have led to this node and drop coefficients that are zero as a consequence of this condition. The result is then a (modified) *Gröbner system* in the sense of [Weispfenning 1992].

A related approach appears in [Kapur 1995a]. Here conditions on the parameters are attached to individual polynomials, resulting in *constrained polynomials*. Parametric Gröbner bases are then finite sets of constrained polynomials. Using this approach the characteristic set construction can also be modified in such a way that it remains stable under specializations [Kapur 1995a].

Once a comprehensive Gröbner basis G has been computed from F , the structure of the variety $V_C(F_a)$ can easily and uniformly be obtained from G depending on the complex values a of the parameters U . In particular there are Boolean combinations of polynomial equations in the parameters that yield necessary and sufficient conditions for the following properties: $V_C(F_a) = \emptyset$, $|V_C(F_a)| = d$, $\dim(V_C(F_a)) = d$ for a given natural number d .

The corresponding problem concerning the structure of $V_R(F_a)$ is much more subtle. Due to the presence of parameters, it is in practice impossible to reduce the problem to the univariate case as in the non-parametric case, since this would involve too many different Gröbner basis calculations.

Progress has, however, been made recently in an important special case, viz. when $V_C(F_a)$ is finite for all relevant values of a . In this case the construction of comprehensive Gröbner bases can be combined with the powerful multivariate Sturm theorem mentioned in Section 4.1 [Becker and Wörmann 1994, Pedersen et al. 1993] to obtain the structure of $V_R(F_a)$ depending on the relevant values of a [Weispfenning 1998]. One proceeds as follows: Using a comprehensive Gröbner basis of $\text{Id}(F)$, one computes \mathbb{Q} -linear bases and structure constants for the finitely many residue class rings $\mathbb{Q}[x_1, \dots, x_n]/(F_a)$ that may occur for all values of the parameters. In each case one computes a certain quadratic form on the respective \mathbb{Q} -linear space. Then the number of real zeros of F in each case is determined by the signature of the corresponding quadratic form in much the same way as in the univariate case described in Section 4.1. For any fixed $k \in \mathbb{N}$ one can write down a Boolean combination of polynomial inequalities in the parameters U expressing the fact that this signature equals k . In this way one gets conditions on the parameters that express the fact that $|V_R(F_a)| = d$. Unfortunately these conditions tend to become very long for large dimensions of the \mathbb{Q} -linear space; so applications so far are limited. Again as in the non-parametric case, side conditions can in principle be incorporated into the method [Dolzmann 1994].

4.3.2. Resultants

Consider a finite set $F \subseteq \mathbb{Q}[U_1, \dots, U_m][x_1, \dots, x_n]$ of parametric polynomials in x_1, \dots, x_n . We have seen above that there is a Boolean combination of polynomial equations in the parameters that are necessary and sufficient conditions for the property $V_C(F_a) \neq \emptyset$. In general, such a Boolean combination cannot be replaced by a conjunction of polynomial equations in the parameters, as the simple example of the polynomial $f(U, x) = Ux - 1$ shows: Here $V_C(f(a, x)) \neq \emptyset$ iff $a \neq 0$. In the special case of testing a system of homogeneous parametric polynomials for a common non-trivial zero, there is indeed a test consisting of a conjunction of polynomial equations in the parameters. The polynomials appearing in such a conjunction are known as a *resultant system* for F [van der Waerden 1940, Section 78].

For an inhomogeneous set of parametric polynomials

$$F \subseteq \mathbb{F}[U_1, \dots, U_m][x_1, \dots, x_n]$$

one may pass to the homogenized system $F^* \subseteq \mathbb{F}[U_1, \dots, U_m][x_0, x_1, \dots, x_n]$ by replacing each $f \in F$ by $f^* = x_0^{\deg(f)} f(x_1/x_0, \dots, x_n/x_0)$. Then a resultant system for F^* constitutes a necessary, but in general not a sufficient criterion for $V_C(F_a) \neq \emptyset$.

A special situation occurs if we take F as the most general set of n homogeneous polynomials f_1, \dots, f_n in n variables x_1, \dots, x_n of fixed degrees d_1, \dots, d_n . Then the parameters are the indeterminate coefficients of these polynomials. We call such a system F *generic*. For the case of a generic system F , there exists a resultant system for F consisting of a single polynomial R in the parameters. If R is taken of minimal degree, then it is uniquely determined up to a constant factor. This polynomial R is called the *resultant* of F [van der Waerden 1940, Section 78] and [Cox et al. 1998]), and is denoted by $\text{Res}_{d_1, \dots, d_n}(f_1, \dots, f_n)$. In the special case, where $n = 2$ and F consists of two homogeneous generic polynomials f_1, f_2 of degree d_1, d_2 , respectively, then $\text{Res}_{d_1, d_2}(f_1, f_2)$ is the well-known *Sylvester resultant*, given by the determinant of the Sylvester-matrix of the coefficients of f_1, f_2 . This resultant is also known as the Sylvester-resultant of the univariate polynomials $f_1(x, 1), f_2(x, 1)$ obtained from f_1, f_2 by dehomogenization.

Due to the very large number of coefficients in a generic system $F = \{f_1, \dots, f_n\}$, the computation of $\text{Res}_{d_1, \dots, d_n}(f_1, \dots, f_n)$ is a major problem of elimination theory. Macauley has given two representations of the resultant, one as a gcd of certain determinants and another as a quotient of two determinants, both having coefficients from F as entries. The first is impractical for computations, and the second approach may fail if the denominator vanishes; moreover for non-generic systems extraneous factors may occur in the second approach [Kapur 1995b]. For ways to circumvent these difficulties see [Canny and Manocha 1993].

Applications of the resultant (for inhomogeneous systems) include the computation of all complex zeros of systems of n polynomials in n variables with finitely many complex zeros, either via the formation of a *u-resultant* or by hiding one variables in the ground field [van der Waerden 1940, Cox et al. 1998].

The complexity of the resultant that is due to dealing with dense polynomials of given degrees is circumvented by *sparse resultants* and more generally *mixed sparse*

resultants [Canny and Emiris 1993, Cox et al. 1998]. Here one restricts attention to sparse multivariate polynomials with specified sets of admissible exponent vectors in the monomials. The polynomials may be inhomogeneous. Call an n -tuple c of complex numbers strongly non-trivial, if all components of c are non-zero. The vanishing of the the mixed sparse resultants of a finite (Laurent) polynomial system F with respect to fixed sets of exponent vectors in \mathbb{Z}^n is then a necessary condition for F to have a common strongly non-trivial zero. It is also sufficient for F to have a common zero in a certain (possibly larger) toric variety V that can be computed from the sets of exponent vectors. A possible way of computing the mixed sparse resultant is as the gcd of certain determinants associated with mixed volumes of tuples of polytopes [Canny and Emiris 1993, Emiris and Canny 1995].

The resultants for homogeneous systems considered above are a special case of sparse resultants, where the set of exponents vectors is given by a bound on the sum of the entries. Another special case are *Dixon resultants*, where the set of exponents vectors is given by individual bounds for each entry. They have been studied in great detail by Kapur and his group [Kapur, Saxena and Yang 1994, Kapur 1995b, Kapur 1998].

4.4. Complex and real quantifier elimination

The first-order theory T_F of fields is expressed in the language L_F of fields having 0, 1 as constants, $+$, $-$, \cdot as binary operations, and equality $=$ as binary relation. For the first-order theory T_{OF} of ordered fields the corresponding language L_{OF} of ordered fields has in addition order $<$ as a binary relation. The axioms for these theories are well-known. So *terms* in L_F or L_{OF} can be written in the form of polynomials with integer coefficients. Atomic formulas in L_F are equations between such terms; in L_{OF} atomic formulas include also inequalities. Formally, integer coefficients of variables and the constant 1 are written in unary notation.

The language L_F is rich enough to express most basic facts in algebraic geometry about zero-sets of polynomial systems. Even more expressive and important for application problems is the language L_{OF} . The sets definable in \mathbb{R} by a L_{OF} -formula are called *semi-algebraic sets*. Among the application areas of semi-algebraic sets are: Geometric theorem proving, computer-aided design and solid modeling, robot kinematics and motion planning, non-linear optimization problems, simulation of technical networks [Chou 1988, Sturm 1997, Sturm and Weispfenning 1997, Sturm and Weispfenning 1998, Weispfenning 1994, Weispfenning 1997c, Dolzmann, Sturm and Weispfenning 1998a, Dolzmann, Sturm and Weispfenning 1998b]. For these applications quantifier elimination is a very powerful tool.

The existence of effective quantifier elimination procedures both for the theory of complex numbers in L_F and the theory of real numbers in L_{OF} is due to Tarski [1948]. An analysis of the properties required in the course of these quantifier elimination procedures provides easily explicit first-order axioms for the theory of \mathbb{C} and of \mathbb{R} . The theory of \mathbb{C} can be axiomatized by first-order axioms for algebraically closed fields of characteristic zero [Tarski 1948]; in fact quantifier elimination holds

for algebraically closed fields of arbitrary characteristic. The theory of \mathbb{R} can be axiomatized by first-order axioms for real closed fields [Tarski 1948].

For applications, the complexity of real or complex quantifier elimination is of decisive importance. Unfortunately, the asymptotical worst-case complexity is doubly exponential [Weispfenning 1988, Davenport and Heintz 1988], compare Section 2.4.2. For practical applications, the situation is not quite as bad. Here we encounter as a rule only formulas with a fixed number of quantifier blocks in its prefix. Under this restriction the lower bound of the worst-case complexity of real or complex quantifier elimination is singly exponential.

Matching upper bounds for the complexity of real quantifier elimination have been proved in [Renegar 1992a, Renegar 1992b, Renegar 1992c], and in a more refined version in [Basu, Pollack and Roy 1998], compare also [Grigoriev 1988, Heintz, Roy and Solernó 1990] for the complexity of the real decision problem. Implementation of these asymptotically optimal algorithms is still at a very early stage [Roy 1996, Rouillier 1996, González-Vega, Roy, Rouillier and Trujillo 1998].

In the following we list some quantifier elimination methods that have been implemented and proven to be useful in applications. They may be specialized or theoretically less efficient in the worst case. For complex elimination a classical method by A. Seidenberg [Seidenberg 1956a] has been refined and implemented in [Wang 1993b, Wang 1993a, Wang 1995]. Another method [Weispfenning 1992, Weispfenning 1998] is based on the construction of *comprehensive Gröbner bases*. As explained in Section 4.3, these bases provide a test for the solvability of a conjunction of polynomial equations with parametric coefficients in \mathbb{C} . This test consists in a Boolean combination of polynomial equations in the parameters, in other words in a quantifier-free formula in these parameters. So we have a complex quantifier elimination for existential formulas of the form

$$\exists x_1 \dots \exists x_n (\bigwedge f_i(x, u) = 0), \quad (4.9)$$

where $u = u_1 \dots u_m$ is an m -tuple of parameters.

Quantifier elimination for an arbitrary existential formula is reduced to this case by formation of a disjunctive normal form and the elimination of negated polynomial equations by the Rabinovich trick (compare Section 1.3):

$$f(x_1 \dots x_n) \neq 0 \iff \exists y (1 - y f(x_1 \dots x_n) = 0) \quad (4.10)$$

This method is implemented in the computer algebra systems MAS [Kredel 1993, Pesch 1994] and REDUCE by A. Dolzmann, W. Neun and T. Sturm (see <http://www.fmi.uni-passau.de/~reduce/cgb/>). An analogous approach for real quantifier elimination is based on parametric multivariate real root counting as described in the previous section [Weispfenning 1998]. It is much more complex and is also implemented in MAS [Lippold 1993, Dolzmann 1994].

The oldest and most comprehensive implemented real quantifier elimination method is quantifier elimination by *partial cylindrical algebraic decomposition* [Collins 1975, Collins 1998, Collins and Hong 1991]. It is based on a cylindrical

decomposition of the variable space of the given formula into finitely many connected semi-algebraic cells in such a way that all polynomials in the input formula are sign invariant on each cell. It includes special strategies for formulas of special types [Hong 1998, Brown 1998].

A special real quantifier elimination system for input formulas with polynomials of low degree is implemented in the REDLOG-package of REDUCE [Dolzmann and Sturm 1996, Dolzmann and Sturm 1997, Dolzmann, Sturm and Weispfenning 1998b]. It is based on the virtual substitution of test points described in Sections 2.2.6 and 4.7.1. This approach is particularly successful for input formulas with many variables and at most quadratic polynomials [Weispfenning 1997b, Weispfenning 1997c, Dolzmann, Sturm and Weispfenning 1998a, Dolzmann, Sturm and Weispfenning 1998b].

Another special real quantifier elimination system for input formula involving only a single quantifier and a single polynomial is described in [González-Vega 1998]; it is based on parametric Sturm-Habicht sequences.

4.5. Further Topics

4.5.1. Quadratic systems of inequalities and quadratic programming.

Quadratic programming is concerned with the minimization of a quadratic objective function $q(x) = x^T Hx + c^T x$ subject to a system $Ax \geq b$ of linear constraints. The method of virtual substitution of test points described in Section 2.2.6 can be extended to Boolean combinations of quadratic inequalities [Weispfenning 1997b]. Here successive elimination of variables by virtual substitution of test points may lead to an increase of degrees beyond two. This situation can sometimes be overcome by implicit factorization of polynomials [Dolzmann, Sturm and Weispfenning 1998a]; otherwise it can lead to an incomplete quantifier elimination. If, however, the constraints except for one quadratic constraint are all linear, this feature will persist throughout successive elimination of variables by virtual substitution of test points. The reason is the following: The test points used for the elimination of a variable x all come from the linear constraints or from a zero of the partial derivative of the objective function with respect x . Hence they are linear polynomials in the remaining variables. Analogous remarks apply to *hyperbolic programming*, i.e. the minimization of a rational linear objective function with respect to linear constraints. The method can also be applied to the corresponding parametric problems.

Under the additional hypothesis that the matrix H is positive definite, the methods of convex programming can be applied yielding Karush–Kuhn–Tucker conditions that lead to a solution by an extension of the Simplex algorithm for linear programming [Peressini, Sullivan and Uhl 1988].

Quadratic programming can be viewed as a decision problem, namely the solvability of a system $Ax \geq b$ of linear constraints together with a quadratic constraint $q(x) \leq c$ derived from the objective function $q(x)$ and a parameter c . As such it is NP-complete [Vavasis 1990]. Indeed, the Boolean satisfiability problem SAT

is easily coded as a quadratic programming problem. On the other hand, as the elimination argument above suggests, satisfiability of the quadratic programming problem can be decided by testing the solvability of certain systems of linear equations derived from the linear constraints and the partial derivatives of $q(x)$. Since the size of the coefficients of these systems is polynomial in the data, this shows that quadratic programming is in NP.

Quadratic constraints in constraint logic programming have been studied in [Pesant and Boyer 1994, Pesant 1995].

4.5.2. Semidefinite programming.

Semidefinite programming is an extension of linear programming where the componentwise inequalities between vectors are replaced by matrix inequalities, or, equivalently the first orthant is replaced by the cone of positive semi-definite matrices. A *semidefinite optimization problem* has the form

$$\min\{c^T x \mid F(x) \succeq 0, x \in \mathbb{R}^n\}, \text{ with } F(x) = F_0 + \sum_{j=1}^n x_j F_j. \quad (4.11)$$

Here F_0, F_1, \dots, F_m are symmetric matrices in $\mathbb{R}^{m \times m}$ and $F(x) \succeq 0$ means that $F(x)$ is positive semidefinite. Such a constraint is nonlinear, but convex; therefore semidefinite programs are convex optimization problems. Semidefinite programming unifies several standard problems, e.g. linear programming and convex quadratically constrained quadratic programming. Although semidefinite programs are much more general than linear programs, they are not much harder to solve. Most interior-point methods for linear programming have been generalized to semidefinite programming. Like in linear programming, these methods have worst-case polynomial complexity and perform well in practice, see [Vandenberghe and Boyd 1996] for an overview.

5. Non-linear diophantine constraints

5.1. Hilbert's Tenth Problem

Non-linear diophantine equations have a long tradition in mathematics, which can be traced back at least to the Greek mathematician Diophantus of Alexandria. Pierre de Fermat, when reading Diophantus' book *Arithmetica*, founded the study of what we call now Diophantine equations. A *diophantine equation* is an equation between polynomials in several variables which has to be solved over the integer numbers. The most famous example is Fermat's equation

$$x^n + y^n = z^n. \quad (5.1)$$

Fermat claimed that for $n \geq 3$ this equation has only the trivial solutions $(a, 0, a)$ and $(0, a, a)$, where a is an arbitrary integer. Only in 1994, this was finally proven by Wiles [1995] and Taylor and Wiles [1995].

In 1900 at the International Congress of Mathematicians in Paris, David Hilbert presented twenty-three challenge problems for the upcoming century. *Hilbert's Tenth Problem* [Matiyasevich 1993] was to find a general method to determine the solvability of a Diophantine equation:

Given a diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.

Only in 1970, this problem was finally proven to be undecidable by Matiyasevich. Let us call a set $D \subseteq \mathbb{Z}^n$ *diophantine* if there exists a polynomial $p \in \mathbb{Z}[x_1, \dots, x_n, y_1, \dots, y_m]$ such that

$$D = \{(a_1, \dots, a_n) \in \mathbb{Z}^n \mid \exists (b_1, \dots, b_m) \in \mathbb{Z}^m \\ p(a_1, \dots, a_n, b_1, \dots, b_m) = 0\}. \quad (5.2)$$

Based on previous work on exponential diophantine equations by Davis, Putnam and Robinson [1961], Matiyasevich showed that every recursively enumerable set $S \subseteq \mathbb{Z}^n$ is diophantine. Since there exist recursively enumerable sets that are not recursive, this implies:

5.1. THEOREM (Matiyasevich 70). *Hilbert's Tenth Problem is undecidable.*

Note that the intersection and union of two diophantine sets is diophantine, since $p = 0 \wedge q = 0$ is equivalent to $p^2 + q^2 = 0$. Moreover, $p = 0 \vee q = 0$ is equivalent to $p \cdot q = 0$. The set of non-negative integers is diophantine by Lagrange's theorem which states that an integer is non-negative if and only if it is the sum of four integer squares. Finally, the set of nonzero integers is diophantine because an integer x is nonzero if and only if there exists a non-negative integer y such that $x = y + 1$ or $x = -y - 1$. This implies that the solvability of systems of diophantine equations, inequalities, and disequalities is equivalent to the solvability of a single diophantine equation.

Matiyasevich's theorem is a very strong result. For example, it implies that there exists a polynomial $p \in \mathbb{Z}[x, y_1, \dots, y_m]$ such that $a \in \mathbb{Z}$ is a prime number iff the diophantine equation $p(a, y_1, \dots, y_m) = 0$ has an integer solution.

Various results have been obtained to characterize the boundary between decidable and undecidable problems in terms of the degree of the polynomial and the number of variables. These can be summarized as follows [Davis, Matiyasevich and Robinson 1976, Tung 1987, Matiyasevich 1993, Mazur 1994, Pheidas 1994]:

- Hilbert's Tenth Problem over \mathbb{N} for a single quadratic diophantine equation in arbitrarily many variables is decidable [Siegel 1972, Grunewald and Segal 1981]. It is undecidable for systems of quadratic equations.
- Hilbert's Tenth Problem over \mathbb{N} for cubic equations is open.
- Hilbert's Tenth Problem over \mathbb{N} for diophantine equations of degree 4 is undecidable.
- Hilbert's Tenth Problem over \mathbb{N} or \mathbb{Z} for systems of equations in one variable is decidable.

- Hilbert's Tenth Problem over \mathbb{N} for equations in 2 to 8 variables is open.
- Hilbert's Tenth Problem over \mathbb{N} for equations in 9 or more variables is undecidable.
- Hilbert's Tenth Problem over \mathbb{Z} for equations in 27 or more variables is undecidable.
- Hilbert's Tenth Problem for solving diophantine equations over the *rational* numbers is a major open problem in the area.

Several authors applied Hilbert's Tenth problem to prove the undecidability of unification problems, see e.g. [Goldfarb 1981, Bockmayr 1987, Siekmann and Szabó 1989].

5.2. Positive results

5.2.1. Effective solution of diophantine equations in two variables

Even in decidable cases, the effective resolution of non-linear diophantine equations is very hard [Smart 1998].

Manders and Adleman [1978] showed that deciding the solvability of diophantine equations of the form

$$ax^2 + by = c, \quad a, b, c \in \mathbb{N} \quad (5.3)$$

over the natural numbers is NP-complete.

In a fundamental contribution, Baker [1968] developed a non-deterministic exponential-time algorithm for deciding the solvability of the *Thue equation*

$$f(x, y) = m, \quad (5.4)$$

where $f(x, y)$ is a homogeneous polynomial of degree ≥ 3 irreducible over the rationals. Baker's method extends to various other equations in two unknowns. The first systematic algorithm for solving Thue's equation was developed by Tzanakis and Weger [1989]. Recent work on Thue's equation includes [Bilu and Hanrot 1996, Bilu and Hanrot 1999], who studied also *superelliptic* diophantine equations of the form

$$ay^p = f(x), \quad (5.5)$$

with $p \geq 3$, a a positive integer, and $f(x) \in \mathbb{Z}[x]$ a separable polynomial of degree $n \geq 2$ [Bilu and Hanrot 1998].

5.2.2. Non-linear integer programming over convex sets

Porkolab and Khachiyan [1997] study integral points in convex subsets of \mathbb{R}^n that are defined by polynomial equations and inequalities. They derive an upper bound for the size of some integral solution - if there is one - and show how this yields a polynomial-time algorithm for these systems in fixed dimension. This result generalizes Lenstra's theorem on the polynomial-time solvability of integer linear programming in fixed dimension.

5.2.3. Non-linear 0-1 programming

Non-linear constraints in 0-1 variables have been studied in the optimization literature. There are four major approaches: linearization, algebraic methods, enumerative methods, and cutting planes, see [Hansen, Jaumard and Mathon 1993] for an overview.

5.3. Decision problems for first-order theories of the integers

The algorithmic unsolvability of Hilbert's Tenth Problem constitutes an enormous strengthening of Gödel's undecidability result [Gödel 1931]. Let T denote the first-order theory of the integers in the language L of rings having 0, 1 as constants, $+$, $-$, \cdot as binary operations, and equality as binary relation. Then Gödel's undecidability theorem says (among other things) that T is undecidable, i.e. there is no algorithm that decides upon input of a closed L -formula φ , whether or not φ holds in \mathbb{Z} . The algorithmic unsolvability of Hilbert's Tenth Problem implies that even the positive existential theory of the integers in L is undecidable: Call an L -formula positive existential if it is existential and contains no negation. Then there is no algorithm that decides upon input of a closed positive existential L -formula φ , whether or not φ holds in \mathbb{Z} .

In fact this result holds even if the inputs are restricted to closed positive existential L -formulas in which all terms are at most quadratic polynomials. To see this, one observes that by introducing new existentially quantified variables the degree of terms can be reduced to two or less. Consider e.g. the equation $x^2yz - 1 = 0$. Then any integer solution of this equation can be expanded to an integer solution of the system $uv - 1 = 0$, $u = yz$, $v = x^2$. By way of contrast, a restriction to linear polynomials (even when order is included in the language) leads to a decidable theory, see Section 3.8. A few restricted non-linear fragments of T turn out to be decidable [Weispfenning 1990a]. On the other hand, the linear theory of the integers with the divisibility relation is undecidable [Lipshitz 1978], since multiplication of integers is definable in this theory as follows: $x \cdot y = z \iff 2z = (x+y)^2 - x^2 - y^2$ and $y = x^2 \iff y + x = \text{lcm}(x, x+1)$. The main result of [Lipshitz 1978] shows, however, that the positive existential fragment of this theory is again decidable.

Acknowledgments

We are grateful to the second reader Michael Maher whose detailed comments helped a lot to improve this chapter. Thanks also to Eric Domenjoud, Fritz Eisenbrand, Guillaume Hanrot, Thomas Kasper, Claude Kirchner, and Nicolai Pissaruk for various useful remarks.

Bibliography

- AARDAL K., HURKENS C. AND LENSTRA A. K. [1998a], Solving a linear diophantine equation with lower and upper bounds on the variables, in 'Integer Programming and Combinatorial Optimization, IPCO'98', Springer, LNCS 1412, pp. 229 – 242.
- AARDAL K., HURKENS C. AND LENSTRA A. K. [1998b], Solving a system of diophantine equations with lower and upper bounds on the variables, Technical Report UU-CS-1998-36, Department of Computer Science, Utrecht University.
- AARDAL K., WEISMANTEL R. AND WOLSEY L. A. [1999], Non-standard approaches to integer programming, Technical Report UU-CS-1999-41, Department of Computer Science, Utrecht University.
- ABDULRAB H. AND MAKSIMENKO M. [1995], General solution of systems of linear diophantine equations and inequations, in 'Rewriting Techniques and Applications, RTA'95, Kaiserslautern', Springer, LNCS 914, pp. 339 – 351.
- ABDULRAB H. AND PÉCUCHE J.-P. [1989], Solving systems of linear diophantine equations and word equations, in 'Rewriting Techniques and Applications, RTA'89, Chapel Hill', Springer, LNCS 355, pp. 530 – 532.
- ADI M. AND KIRCHNER C. [1992], 'AC-Unification Race: The System Solving Approach, Implementation and Benchmarks', *J. Symbolic Computation* **14**(1), 51–70.
- AJILI F. [1994], Etude de la résolution de contraintes diophantiennes linéaires sur les entiers naturels, Mémoire de DEA, Univ. Henri Poincaré, Nancy, France.
- AJILI F. [1998], Contraintes diophantiennes linéaires: résolution et coopération inter-résolveurs, PhD thesis, Univ. Henri Poincaré, Nancy, France.
- AJILI F. AND CONTEJEAN E. [1997], 'Avoiding slack variables in the solving of linear diophantine equations and inequations', *Theoretical Computer Science* **173**, 183 – 208.
- AJILI F. AND LOCK H. C. R. [1998], Integrating constraint propagation in complete solving of linear diophantine systems, in 'Principles of Declarative Programming, ALP/PLILP'98, Pisa', Springer, LNCS 1490, pp. 463 – 480.
- AJTAI M. [1998], The shortest vector problem in l_2 is NP-hard for randomized reduction, in '30th ACM Symposium on the Theory of Computing, STOC'98, Dallas', pp. 10 – 19.
- ANAI H. AND WEISPFENNING V. [2000], Deciding linear-trigonometric problems, in C. Traverso, ed., 'International Symposium on Symbolic and Algebraic Computation, ISSAC 2000', ACM-Press, pp. 14–22.
- APEL J. [1995], 'A Gröbner approach to involutive bases', *J. Symb. Comp.* **19**, 441–457.
- APEL J. [1998], 'The theory of involutive divisions and an application to Hilbert function computations', *J. Symb. Comp.* **25**, 683–704.
- ARMANDO A., MELIS E. AND RANISE S. [1998], Constraint solving in logic programming and in automated deduction: a comparison, in 'Artificial Intelligence: Methodology, Systems, Applications, AIMSA'98', Springer, LNCS 1480, pp. 28–38.
- ARMANDO A. AND RANISE S. [1998], From integrated reasoning specialists to "plug-and-play" reasoning components, in 'Artificial Intelligence and Symbolic Computation, AISC'98', Springer, LNCS 1476, pp. 42–54.
- ARORA S., BABAI L., STERN J. AND SWEEDYK Z. [1993], The hardness of approximate optima in lattices, codes, and systems of linear equations, in 'Foundations of Computer Science FOCS'93, Palo Alto'.
- AZULAY D.-O. AND PIQUE J.-F. [1998], Optimized Q-pivot for exact linear solvers, in 'Principles and Practice of Constraint Programming, CP'98, Pisa', Springer, LNCS 1520, pp. 55 – 71.
- BAADER F. AND SNYDER W. [2001], Unification theory, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 8, pp. 445–532.
- BACHMAIR L. AND GANZINGER H. [2001], Resolution theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19–99.

- BAKER A. [1968], 'Contributions to the theory of diophantine equations. I/II', *Philos. Trans. Roy. Soc. London, Ser. A* **263**, 173 – 208.
- BALAS E. [1979], 'Disjunctive programming', *Annals of Discrete Mathematics* **5**, 3 – 51.
- BALAS E. [1998], 'Disjunctive programming: Properties of the convex hull of feasible points', *Discrete Applied Mathematics* **89**(1-3), 3 – 44.
- BALAS E., CERIA S. AND CORNUÉJOLS G. [1993], 'A lift-and-project cutting plane algorithm for mixed 0-1 programs', *Mathematical Programming* **58**, 295 – 324.
- BALAS E., CERIA S. AND CORNUÉJOLS G. [1996], 'Mixed 0-1 programming by lift-and-project in a branch-and-cut framework', *Management Science* **42**(9), 1229 – 1246.
- BALAS E., CERIA S., CORNUÉJOLS G. AND NATRAJ N. R. [1996], 'Gomory cuts revisited', *Operations Research Letters* **19**.
- BALAS E. AND PERREGAARD M. [2000], A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming, Management Science Research Report # MSRR-631, GSIA, Carnegie Mellon University.
- BAREISS E. H. [1968], 'Sylvester's identity and multistep integer-preserving Gaussian elimination', *Mathematics of Computation* **22**, 565 – 578.
- BARTH P. [1996], *Logic-based 0-1 constraint programming*, Operations Research/Computer Science Interfaces Series, Kluwer.
- BARTH P. AND BOCKMAYR A. [1995], Finite domain and cutting plane techniques in CLP(\mathcal{PB}), in L. Sterling, ed., 'Logic Programming. 12th International Conference, ICLP'95, Kanagawa, Japan', MIT Press, pp. 133 – 147.
- BARTH P. AND BOCKMAYR A. [1998], 'Modelling discrete optimisation problems in constraint logic programming', *Annals of Operations Research* **81**, 467–496.
- BASU S., POLLACK R. AND ROY M.-F. [1998], A new algorithm to find a point in every cell defined by a family of polynomials, in B. Caviness and J. Johnson, eds, 'Quantifier Elimination and Cylindrical Algebraic Decomposition', Springer, Wien, New York, pp. 341–350.
- BECKER E. AND WÖRMANN T. [1994], On the trace formula for quadratic forms, in W. B. Jacob, T.-Y. Lam and R. O. Robson, eds, 'Recent Advances in Real Algebraic Geometry and Quadratic Forms', Vol. 155 of *Contemporary Mathematics*, American Mathematical Society, Providence, Rhode Island, pp. 271–291. Proceedings of the RAGSQUAD Year, Berkeley, 1990–1991.
- BECKER T., WEISPFENNING V. AND KREDEL H. [1993], *Gröbner Bases, a Computational Approach to Commutative Algebra*, Vol. 141 of *Graduate Texts in Mathematics*, Springer, New York.
- BEN-OR M., KOZEN D. AND REIF J. [1986], 'The complexity of elementary algebra and geometry', *Journal of Computer and System Sciences* **32**, 251–264.
- BERMAN L. [1980], 'The complexity of logical theories', *Theoretical Computer Science* **11**, 71–77.
- BERTOLI P. G., CALMET J., GIUNCHIGLIA F. AND HOMANN K. [1998], Specification and integration of theorem provers and computer algebra systems, in 'Artificial Intelligence and Symbolic Computation, AISC'98', Springer, LNCS 1476, pp. 94–106.
- BERTSIMAS D. AND TSITSIKLIS J. N. [1997], *Introduction to linear optimization*, Athena Scientific, Belmont, Mass.
- BILU Y. AND HANROT G. [1996], 'Solving Thue equations of high degree', *J. Number Th.* **60**, 373–392.
- BILU Y. AND HANROT G. [1998], 'Solving superelliptic diophantine equations by Baker's method', *Compositio Math.* **112**, 273–312.
- BILU Y. AND HANROT G. [1999], 'Thue equations with composite fields', *Acta Arithmetica* **88**(4), 311–326.
- BJØRNER N. S., STICKEL M. E. AND URIBE T. E. [1997], A practical integration of first-order reasoning and decision procedures, in '14th International Conference on Automated Deduction, CADE-14, Townsville', Springer, LNCS 1249.

- BLAIR C. AND JEROSLOW R. G. [1978], 'A converse for disjunctive constraints', *Journal of Optimization Theory and Applications* **25**, 195 – 206.
- BLAND R. [1977], 'New finite pivoting rules for the simplex method', *Mathematics of Operations Research* **2**, 103 – 107.
- BLEDSE W. W. [1975], A new method for proving certain Presburger formulas, in 'Fourth Int. Joint Conf. on Artificial Intelligence, IJCAI'75, Tbilisi'.
- BLÖMER J. AND SEIFERT J.-P. [1999], On the complexity of computing short linearly independent vectors and short bases in a lattice, in 'ACM Symposium on Theory of computing, STOC'99, Atlanta, GA', pp. 711 – 720.
- BOCHNAK J., COSTE M. AND ROY M.-F. [1987], *Géométrie algébrique réelle*, Springer, Berlin, Heidelberg, New York.
- BOCKMAYR A. [1987], 'A note on a canonical theory with undecidable unification and matching problem', *Journal of Automated Reasoning* **3**, 379 – 381.
- BOCKMAYR A. [1993], Logic programming with pseudo-Boolean constraints, in F. Benhamou and A. Colmerauer, eds, 'Constraint Logic Programming. Selected Research', MIT Press, chapter 18, pp. 327 – 350.
- BOCKMAYR A. [1995], Solving pseudo-Boolean constraints, in 'Constraint Programming: Basics and Trends', Springer, LNCS 910, pp. 22 – 38.
- BOCKMAYR A. AND EISENBRAND F. [1999], Cutting planes and the elementary closure in fixed dimension, Research Report MPI-I-99-2-008, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany. To appear in *Mathematics of Operations Research*.
- BOCKMAYR A., EISENBRAND F., HARTMANN M. AND SCHULZ A. S. [1999], 'On the Chvátal rank of polytopes in the 0/1 cube', *Discrete Applied Mathematics* **98**, 21 – 27.
- BOCKMAYR A. AND KASPER T. [1998], 'Branch-and-infer: A unifying framework for integer and finite domain constraint programming', *INFORMS J. Computing* **10**(3), 287 – 300.
- BORGWARDT K.-H. [1982], 'Some distribution-independent results about the asymptotic order of the average number of pivot steps of the simplex method', *Mathematics of Operations Research* **7**, 441 – 462.
- BOUDET A. AND COMON H. [1996], Diophantine equations, Presburger arithmetic and finite automata, in 'Trees in Algebra and Programming, CAAP'96, Linköping, Sweden', Springer, LNCS 1059, pp. 30 – 44.
- BOYER R. S. AND MOORE J. S. [1988], Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic, in J. E. Hayes, D. Michie and J. Richards, eds, 'Machine Intelligence 11: Logic and the acquisition of knowledge', Clarendon Press, chapter 5, pp. 83–124.
- BROWN C. [1998], Simplification of truth-invariant cylindrical algebraic decomposition, in 'International Symposium on Symbolic and Algebraic Computation, ISSAC'98', ACM Press, pp. 295–301.
- BRUNS W., GUBELADZE J., HENK M., MARTIN A. AND WEISMANTEL R. [1999], 'A counterexample to an integer analogue of Carathéodory's theorem', *J. Reine Angew. Math.* **510**, 179–185.
- BÜNDGEN R. [1996], 'Buchberger's algorithm: the term rewriter's point of view', *Theor. Comp. Science* **159**(2), 143–190.
- BÜRCKERT H.-J. [1991], *A Resolution principle for a logic with restricted quantifiers*, Springer, LNCS 568.
- BÜRCKERT H.-J., HEROLD A., KAPUR D., SIEKMANN J. H., STICKEL M. E., TEPP M. AND ZHANG H. [1988], 'Opening the AC-unification race', *Journal of Automated Reasoning* **4**(4), 465–474.
- BURG J., STUCKEY P. J., TAI J. C. H. AND YAP R. H. C. [1995], Linear equation solving for constraint logic programming, in 'Logic Programming. 12th International Conference, ICLP'95, Kanagawa, Japan', MIT Press, pp. 33 – 47.
- BÜRGISSE P., CLAUSEN M. AND SHOKROLLAHI M. A. [1997], *Algebraic Complexity Theory*, Vol. 315 of *Grundlehren der Mathematischen Wissenschaften*, Springer.

- BURHENNE K.-D. [1990], Implementierung eines Algorithmus zur Quantorenelimination für lineare reelle Probleme, Diploma thesis, Universität Passau, D-94030 Passau, Germany.
- CANNY J. AND EMIRIS I. [1993], An efficient algorithm for the sparse resultant, in 'Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, AAECC-93', Vol. 263 of *LNCS*, Springer, pp. 89–104.
- CANNY J. AND MANOCHA D. [1993], 'Multipolynomial resultant algorithms', *Journal of Symbolic Computation* 15, 99–122.
- CAPRARA A., FISCHETTI M. AND LETCHFORD A. N. [2000], 'On the separation of maximally violated mod- k cuts', *Mathematical Programming* 87, 37–56.
- ČERNIKOV R. [1963], 'Contraction of finite systems of linear inequalities', *Soviet Mathematics Doklady* 4(5), 1520–1524.
- CHANDRU V. [1993], 'Variable elimination in linear constraints', *The Computer Journal* 36(5), 463 – 472.
- CHANDRU V. AND HOOKER J. N. [1999], *Optimization Methods for Logical Inference*, Wiley.
- CHOU S.-C. [1988], *Mechanical Geometry Theorem Proving*, Mathematics and its applications, D. Reidel Publishing Company, Dordrecht, Boston, Lancaster, Tokyo.
- CHOU T.-W. AND COLLINS G. E. [1982], 'Algorithms for the solution of systems of linear diophantine equations', *SIAM J. Computing* 11(4), 687 – 708.
- CHVÁTAL V. [1973], 'Edmonds polytopes and a hierarchy of combinatorial problems', *Discrete Mathematics* 4, 305 – 337.
- CLAUSEN M. AND FORTENBACHER A. [1989], 'Efficient solution of linear diophantine equations', *Journal of Symbolic Computation* 8, 201–216.
- COHEN E. AND MEGIDDO N. [1994], 'Improved algorithms for linear inequalities with two variables per inequality', *SIAM J. Computing* 23(6), 1313–1347.
- COLLINS G. E. [1975], Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition, in H. Brakhage, ed., 'Automata Theory and Formal Languages. 2nd GI Conference', Vol. 33 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Heidelberg, New York, pp. 134–183.
- COLLINS G. E. [1985], The SAC-2 computer algebra system, in B. F. Caviness, ed., 'EUROCAL '85; Proceedings of the European Conference on Computer Algebra', Vol. 104 of *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, New York, Tokyo, pp. 34–35.
- COLLINS G. E. [1998], Quantifier elimination by cylindrical algebraic decomposition - twenty years of progress, in B. Caviness and J. Johnson, eds, 'Quantifier Elimination and Cylindrical Algebraic Decomposition', Springer, Wien, New York, pp. 8–23.
- COLLINS G. E. AND HONG H. [1991], 'Partial cylindrical algebraic decomposition for quantifier elimination', *Journal of Symbolic Computation* 12(3), 299–328.
- COLMERAUER A. [1987], Introduction to PROLOG III, in '4th Annual ESPRIT Conference, Bruxelles', North Holland.
- COMON H. [1991], Disunification: A survey, in J. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of A. Robinson', MIT Press, chapter 9, pp. 322–359.
- COMON H., DINGBAS M., JOUANNAUD J.-P. AND KIRCHNER C. [1999], 'A methodological view of constraint solving', *Constraints* 4, 337 – 361.
- CONTEJEAN E. AND DEVIE H. [1994], 'An efficient incremental algorithm for solving systems of linear diophantine equations', *Information and Computation* 113, 143 – 172.
- CONTI P. AND TRAVERSO C. [1991], Buchberger algorithm and integer programming, in 'Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, AAECC-9', Springer, LNCS 539.
- COOK W., COULLARD C. R. AND TURÁN G. [1987], 'On the complexity of cutting plane proofs', *Discrete Applied Mathematics* 18, 25 – 38.
- COOK W., GERARDS A. M. H., SCHRIJVER A. AND TARDOS E. [1986], 'Sensitivity theorems in integer linear programming', *Mathematical Programming* 34, 251 – 264.

- COOK W., RUTHERFORD T., SCARF H. E. AND SHALLCROSS D. [1993], 'An implementation of the generalized basis reduction algorithm for integer programming', *ORSA Journal on Computing* 5(2), 206 – 212.
- COOPER D. [1972], 'Theorem-proving in arithmetic without multiplication', *Machine Intelligence* 7, 91–100.
- COPPERSMITH D. AND WINOGRAD S. [1990], 'Matrix multiplication via arithmetic progressions', *Journal of Symbolic Computation* 9(3), 251 – 280.
- COX D., LITTLE J. AND O'SHEA D. [1992], *Ideals, Varieties and Algorithms*, Undergraduate Texts in Mathematics, Springer-Verlag, New York, Berlin, Heidelberg.
- COX D., LITTLE J. AND O'SHEA D. [1998], *Using algebraic geometry*, Graduate Texts in Mathematics, Springer-Verlag, New York, Berlin, Heidelberg.
- DANTZIG G. [1963], *Linear Programming and Extensions*, Princeton Univ. Press, Princeton.
- DANTZIG G. AND EAVES B. [1973], 'Fourier-Motzkin elimination and its dual', *Journal of Combinatorial Theory, Series A* 14, 288–297.
- DAVENPORT J. H. AND HEINTZ J. [1988], 'Real quantifier elimination is doubly exponential', *Journal of Symbolic Computation* 5(1–2), 29–35.
- DAVIS M., MATIYASEVICH Y. AND ROBINSON J. [1976], 'Hilbert's tenth problem. Diophantine equations: Positive aspects of a negative solution', *Proc. Symp. Pure Math.* 28, 323–378.
- DAVIS M., PUTNAM H. AND ROBINSON J. [1961], 'The decision problem for exponential Diophantine equations', *Annals of Mathematics* 74(3).
- DEGTYAREV A. AND VORONKOV A. [2001], Equality reasoning in sequent-based calculi, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 10, pp. 609–704.
- DIKIN I. [1967], 'Iterative solution of problems of linear and quadratic programming', *Soviet Math. Doklady* 8, 674–675.
- DINCBAŞ M., VAN HENTENRYCK P., SIMONIS H., AGGOUN A. AND GRAF T. [1988], The constraint logic programming language CHIP, in 'Fifth Generation Computer Systems, Tokyo, 1988', Springer.
- DINES L. [1919], 'Systems of linear inequalities', *Annals of Mathematics* 20, 191–199.
- DOLZMANN A. [1994], Reelle Quantorenelimination durch parametrisches Zählen von Nullstellen, Diploma thesis, Universität Passau, D-94030 Passau, Germany.
- DOLZMANN A., GLOOR O. AND STURM T. [1998], Approaches to parallel quantifier elimination, in O. Gloor, ed., 'International Symposium on Symbolic and Algebraic Computation, ISSAC 98', ACM Press, Rostock, Germany, pp. 88–95.
- DOLZMANN A. AND STURM T. [1996], Redlog User manual, Technical Report MIP-9616, FMI, Universität Passau, D-94030 Passau, Germany. Edition 1.0 for Version 1.0.
- DOLZMANN A. AND STURM T. [1997], 'Redlog: Computer algebra meets computer logic', *ACM SIGSAM Bulletin* 31(2), 2–9.
- DOLZMANN A., STURM T. AND WEISPFENNING V. [1998a], 'A new approach for automatic theorem proving in real geometry', *Journal of Automated Reasoning* 21(3), 357–380.
- DOLZMANN A., STURM T. AND WEISPFENNING V. [1998b], Real quantifier elimination in practice, in B. H. Matzat, G.-M. Greuel and G. Hiss, eds, 'Algorithmic Algebra and Number Theory', Springer, Berlin, pp. 221–247.
- DOMENJOU E. [1991], Solving systems of linear diophantine equations: An algebraic approach, in 'Mathem. Foundations of Computer Science, Kazimierz Dolny, Poland', Springer, LNCS 520.
- DOMENJOU E. [1992], 'A technical note on AC-unification. the number of minimal unifiers of the equation $\alpha x_1 + \dots + \alpha x_p \doteq_{ac} \beta y_1 + \dots + \beta y_q$ ', *Journal of Automated Reasoning* 8, 39–44.
- DOMENJOU E. AND TOMÁS A. P. [1995], From Elliott-MacMahon to an algorithm for general linear constraints in naturals, in 'Principles and Practice of Constraint Programming - CP'95, Cassis, France', Springer, LNCS 976, pp. 18 – 35.

- DORMICH P. D., KANNAN R. AND TROTTER L. E. [1987], 'Hermite normal form computation using modulo determinant arithmetic', *Mathematics of Operations Research* **12**(1), 50 – 59.
- DOWEK G., HARDIN T. AND KIRCHNER C. [1998], Theorem proving modulo, Rapport de recherche 3400, INRIA. To appear in *J. Autom. Reas.*
- DRIES V. D. L. AND HOLLY J. [1992], 'Quantifier elimination for modules with scalar variables', *Annals of Pure and Applied Logic* **57**, 161–179.
- DUFFIN R. J. [1974], 'On Fourier's analysis of linear inequality systems', *Mathematical Programming Study* **1**, 71–95.
- DURAND A., HERMANN M. AND JUBAN L. [1999], On the complexity of recognizing the Hilbert basis of a linear diophantine system, in '24th International Symposium on Mathematical Foundations of Computer Science, MFCS'99', Springer, LNCS 1672, pp. 92–102.
- EAVES B. AND ROTHBLUM U. [1992], 'Dines-Fourier-Motzkin quantifier elimination and an application of corresponding transfer principles over ordered fields', *Mathematical Programming* **53**, 307–321.
- EDMONDS J. [1967], 'Systems of distinct representatives and linear algebra', *J. Res. Natl. Bur. Stand., Sect. B* **71**, 241–245.
- EISENBRAND F. [1999], 'On the membership problem for the elementary closure of a polyhedron', *Combinatorica* **19**(2), 297–300.
- EISENBRAND F. [2000], Gomory-Chvátal cutting planes and the elementary closure of polyhedra, PhD thesis, Naturwissenschaftlich-Technische Fakultät I, Univ. des Saarlandes.
- EISENBRAND F. AND SCHULZ A. S. [1999], Bounds on the Chvátal rank of polytopes in the 0/1 cube, in 'Integer Programming and Combinatorial Optimization, IPCO 99', Springer, LNCS 1610, pp. 137–150.
- ELLIOTT E. B. [1903], 'On linear homogeneous diophantine equations', *Quarterly Journal of Pure and Applied Mathematics* **34**, 348 – 377.
- EMIRIS I. AND CANNY J. [1995], 'Efficient incremental algorithms for the sparse resultant and the mixed volume', *J. Symbolic Computation* **20**, 117–149.
- FANG X. G. AND HAVAS G. [1997], On the worst-case complexity of integer Gaussian elimination, in 'International Symposium on Symbolic and Algebraic Computation, ISSAC'97, Maui, HI', ACM Press, pp. 28 – 31.
- FERRANTE J. AND RACKOFF C. W. [1979], *The Computational Complexity of Logical Theories*, number 718 in 'Lecture Notes in Mathematics', Springer-Verlag, Berlin.
- FILGUEIRAS M. AND TOMÁS A. P. [1991], Solving linear constraints on finite domains through parsing, in '5th Portuguese Conference on Artificial Intelligence, EPIA'91', Springer, LNCS 541, pp. 1–16.
- FILGUEIRAS M. AND TOMÁS A. P. [1993], Fast methods for solving linear diophantine equations, in '6th Portuguese Conference on AI, EPIA'93', Springer, LNCS 727, pp. 297 – 306.
- FISCHER M. J. AND RABIN M. [1974], 'Super-exponential complexity of Presburger arithmetic', *SIAM-AMS Proceedings* **7**, 27–41.
- FORTENBACHER A. [1993], Algebraische Unifikation (in German), Diplomarbeit, Univ. Karlsruhe.
- FOURIER J. [1826], 'Solution d'une question particulière du calcul des inégalités', *Nouveau Bulletin des Sciences par la Société Philomathique de Paris* pp. 99–100.
- FRANK A. AND TARDOS E. [1987], 'An application of simultaneous diophantine approximation in combinatorial optimization', *Combinatorica* **7**, 49–65.
- FRUMKIN M. A. [1976], 'An algorithm for the reduction of a matrix of integers to triangular form with power complexity of computations (in Russian)', *Ekonom. i mat. Metody* **12**, 173 – 178.
- FÜRER M. [1982], 'The complexity of Presburger arithmetic with bounded quantifier alternation depth', *Theoretical Computer Science* **18**, 105–111.
- GAO X. AND CHOU S. [1991], Computing with parametric equations, in S. Watt, ed., 'International Symposium on Symbolic and Algebraic Computation, ISSAC'91', ACM Press, pp. 122–127.

- GATHEN V. Z. J. AND SIEVEKING M. [1976], Weitere zum Erfüllungsproblem polynomial äquivalente kombinatorische Aufgaben, in E. Specker and V. Strassen, eds, 'Komplexität von Entscheidungsproblemen', Vol. 43 of *LNCS*, Springer.
- GATHEN V. Z. J. AND SIEVEKING M. [1978], 'A bound on solutions of linear integer equalities and inequalities', *Proceedings AMS* **72**, 155–158.
- GEDDES K. O., CZAPOR S. R. AND LABAHN G. [1992], *Algorithms for computer algebra*, Kluwer.
- GERARDS A. M. H. [1990], On cutting planes and matrices, in W. Cook and P. D. Seymour, eds, 'Polyhedral Combinatorics', DIMACS, pp. 29–32.
- GÖDEL K. [1931], 'Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I', *Monatsh. Math. Phys.* **38**, 173–198.
- GOLDFARB W. [1981], 'The undecidability of the second-order unification problem', *Theoretical Computer Science* **13**, 225–230.
- GOMORY R. E. [1958], 'Outline of an algorithm for integer solutions to linear programs', *Bull. AMS* **64**, 275 – 278.
- GONZÁLEZ L., LOMBARDI H., RECIO T. AND ROY M.-F. [1989], Sturm-Habicht sequence, in 'International Symposium on Symbolic and Algebraic Computation, ISSAC89', ACM Press, pp. 136–146.
- GONZÁLEZ L., LOMBARDI H., RECIO T. AND ROY M.-F. [1990], 'Sous-résultants et spécialisation de la suite de Sturm I', *Informatique théorique et applications* **24**(6), 561–588.
- GONZÁLEZ-VEGA [1998], A combinatorial algorithm solving some quantifier elimination problems, in B. Caviness and J. Johnson, eds, 'Quantifier Elimination and Cylindrical Algebraic Decomposition', Springer, Wien, New York, pp. 365–375.
- GONZÁLEZ-VEGA L., RECIO T., LOMBARDI H. AND ROY M.-F. [1998], Sturm-Habicht sequences determinants and real roots of univariate polynomials, in B. Caviness and J. Johnson, eds, 'Quantifier Elimination and Cylindrical Algebraic Decomposition', Springer, Wien, New York, pp. 300–316.
- GONZÁLEZ-VEGA L., ROY M.-F., ROUILLIER F. AND TRUJILLO G. [1998], Symbolic recipes for real solutions, in A. Cohen, H. Cuypers and H. Sterk, eds, 'Some tapas of computer algebra', Springer, Berlin.
- GORDAN P. [1873], 'Über die Auflösung linearer Gleichungen mit reellen Coefficienten', *Mathematische Annalen* **6**, 23 – 28.
- GRÄBE H.-G. [1994], On factorized Gröbner bases, in 'Proceedings of the workshop: "Computer Algebra in Science and Engineering", Bielefeld, Aug'94', World Scientific, Singapore, pp. 77–89.
- GRÄDEL E. [1987], The complexity of subclasses of logical theories, Doctoral dissertation, Basel.
- GRIGORIEV D. [1988], 'Complexity of deciding Tarski algebra', *Journal of Symbolic Computation* **5**(1), 65–108.
- GRÖTSCHEL M., LOVÁSZ L. AND SCHRIJVER A. [1988], *Geometric algorithms and combinatorial optimization*, Vol. 2 of *Algorithms and Combinatorics*, Springer.
- GRUNEWALD F. J. AND SEGAL D. [1981], 'How to solve a quadratic equation in integers', *Math. Proc. Camb. Philos. Soc.* **89**, 1 – 5.
- GUCKENBIEHL T. AND HEROLD A. [1985], Solving linear diophantine equations, Technical Report Memo SEKI-85-IV-KL, Univ. Kaiserslautern.
- GURARI E. [1985], 'Decidable problems for powerful programs', *J. ACM* **32**, 466–483.
- GURARI E. AND IBARRA O. [1979], 'An NP-complete number-theoretic problem', *J. ACM* **26**, 567–581.
- GURARI E. AND IBARRA O. [1981], 'The complexity of the equivalence problem for two characterizations of Presburger sets', *Theoretical Computer Science* **13**, 295–314.
- HAFNER J. L. AND MCCURLEY K. S. [1991], 'Asymptotically fast triangularization of matrices over rings', *SIAM J. Computing* **20**(6), 1068 – 1083.
- HAKEN A. [1985], 'The intractability of resolution', *Theoretical Computer Science* **39**, 297 – 308.

- HAKEN A. AND COOK S. A. [1999], 'An exponential lower bound for the size of monotone real circuits', *Journal of Computer and System Sciences* **58**, 326 – 335.
- HANSEN P., JAUMARD B. AND MATHON V. [1993], 'Constrained nonlinear 0-1 programming', *ORSA Journal on Computing* **5**(2), 97 – 119.
- HAVAS G., HOLT D. F. AND REES S. [1993], 'Recognizing badly presented Z -modules', *Linear Algebra Appl.* **192**, 137 – 163.
- HAVAS G. AND MAJEWSKI B. S. [1994], 'Hermite normal form computation for integer matrices', *Congressus Numerantium* **105**, 87 – 96.
- HAVAS G. AND MAJEWSKI B. S. [1997], 'Integer matrix diagonalization', *Journal of Symbolic Computation* **24**, 399 – 408.
- HAVAS G., MAJEWSKI B. S. AND MATTHWES K. R. [1998], 'Extended GCD and Hermite normal form algorithms via lattice basis reduction', *Experimental Mathematics* **7**(2), 125 – 136.
- HEINTZ J., ROY M.-F. AND SOLERNÓ P. [1990], 'Sur la complexité du principe de Tarski-Seidenberg', *Bull. Soc. Math. France* **118**, 101–126.
- HENK M. [1997], 'Note on shortest and nearest lattice vectors', *Information Processing Letters* **61**, 183 – 188.
- HENK M. AND WEISMANTEL R. [1997], 'The height of minimal Hilbert bases', *Result. Math.* **32**(3-4), 298 – 303.
- HENK M. AND WEISMANTEL R. [2000], 'On minimal solutions of linear diophantine equations', *Contrib. Algebra and Geometry* **41**(1), 49–55.
- HERMANN M., JUBAN L. AND KOLAITIS P. G. [1999], On the complexity of counting the Hilbert basis of a linear diophantine system, in '6th International Conference on Logic for Programming and Automated Reasoning, LPAR'99', Springer, LNCS 1705, pp. 13–32.
- HERMITE C. [1851], 'Sur l'introduction de variables continues dans la théorie des nombres', *J. reine u. angewandte Mathematik* **41**, 191 – 216.
- HILBERT D. [1890], 'Über die Theorie der algebraischen Formen', *Mathematische Annalen* **36**, 473 – 534.
- HOCHBAUM D. AND NAOR J. [1994], 'Simple and fast algorithms for linear and integer programs with two variables per inequality', *SIAM J. Computing* **23**(6), 1179–1192.
- HOLZBAUR C. [1995], OEFAI clip(Q,R) manual, Edition 1.3.3, Technical Report TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna.
- HONG H. [1998], An improvement of the projection operator in cylindrical algebraic decomposition, in B. Caviness and J. Johnson, eds, 'Quantifier Elimination and Cylindrical Algebraic Decomposition', Springer, Wien, New York, pp. 166–173.
- HOOKE J. N. [1988a], 'Generalized resolution and cutting planes', *Annals of Operations Research* **12**, 217 – 239.
- HOOKE J. N. [1988b], 'A quantitative approach to logical inference', *Decision Support Systems* **4**, 45 – 69.
- HOOKE J. N. [1992], 'Generalized resolution for 0-1 linear inequalities', *Annals of Mathematics and Artificial Intelligence* **6**, 271–286.
- HOOKE J. N. [2000], *Logic-based methods for optimization*, John Wiley.
- HUET G. [1972], Constrained Resolution: A Complete Method for Higher Order Logic, PhD thesis, Case Western Reserve University, USA.
- HUET G. [1978], 'An algorithm to generate the basis of solutions to homogeneous linear diophantine equations', *Information Processing Letters* **7**(3), 144 – 147.
- HUYNH T., JOSKOWICZ L., LASSEZ C. AND LASSEZ J. L. [1990], Reasoning about constraints using parametric queries, in K. Nori and C. Veni Madhavan, eds, 'Foundations of Software Technology and Theoretical Computer Science', number 472 in 'LNCS', Springer, pp. 1–20.
- HUYNH T., LASSEZ C. AND LASSEZ J.-L. [1992], 'Practical issues on the projection of polyhedral sets', *Annals of Mathematics and Artificial Intelligence* **6**, 295–316.
- HUYNH T. AND MARRIOTT K. [1995], 'Incremental constraint deletion in systems of linear constraints', *Information Processing Letters* **55**, 111 – 115.

- ILIOPOULOS C. S. [1989a], 'Worst-case complexity bounds on algorithms for computing the canonical structure of finite abelian groups and the Hermite and Smith normal forms of an integer matrix', *SIAM J. Computing* **18**(4), 658 – 669.
- ILIOPOULOS C. S. [1989b], 'Worst-case complexity bounds on algorithms for computing the canonical structure of infinite abelian groups and solving systems of linear diophantine equations', *SIAM J. Computing* **18**(4), 670–678.
- IMBERT J. L. [1993], 'Variable elimination for disequations in generalized linear constraint systems', *The Computer Journal* **36**(5), 473 – 484.
- IMBERT J.-L. [1995a], Fourier's elimination: which to choose, in 'Principles and Practice of Constraint Programming. The Newport Papers', MIT Press, pp. 245 – 268.
- IMBERT J.-L. [1995b], Linear constraint solving in CLP languages, in 'Constraint Programming: Basics and Trends', Springer, LNCS 910, pp. 108 – 127.
- IMBERT J.-L. [1997], Redundancy, variable elimination and linear disequations, in 'International Symposium on Logic Programming, ISLP'94, Ithaca', MIT Press, pp. 139 – 153.
- JAFFAR J. AND LASSEZ J.-L. [1987], Constraint logic programming, in 'Proc. 14th ACM Symp. Principles of Programming Languages', Munich.
- JAFFAR J. AND MAHER M. J. [1994], 'Constraint logic programming: A survey', *Journal of Logic Programming* **19/20**, 503 – 581.
- JAFFAR J., MAHER M. J., STUCKEY P. J. AND YAP R. H. C. [1993], 'Projecting CLP(\mathcal{R}) constraints', *New Generation Computing* **11**, 449–469.
- JAFFAR J., MICHAYLOV S., STUCKEY P. J. AND YAP R. H. C. [1992], 'The CLP(\mathcal{R}) language and system', *ACM Transactions on Programming Languages and Systems* **14**(3), 339–395.
- JANIČIĆ P., BUNDY A. AND GREEN I. [1999], A framework for flexible integration of a class of decision procedures into theorem provers, in 'Automated Deduction, CADE-16, Trento', Springer, LNAI 1632, pp. 127 – 141.
- JEROSLOW R. G. [1977], 'Cutting-plane theory: disjunctive methods', *Annals of Discrete Mathematics* **1**, 293 – 330.
- JOHNSON E. L., NEMHAUSER G. L. AND SAVELSBERGH M. W. P. [2000], 'Progress in linear programming-based algorithms for integer programming: An exposition', *INFORMS J. Computing* **12**, 2 – 23.
- JOUANNAUD J. P. AND KIRCHNER C. [1991], Solving equations in abstract algebras: A rule-based survey of unification, in J. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of A. Robinson', MIT Press, chapter 8, pp. 257–321.
- KANNAN R. AND BACHEM A. [1979], 'Polynomial algorithms for computing the Smith and the Hermite normal forms of an integer matrix', *SIAM J. Computing* **8**(4), 499 – 507.
- KAPPERT M. [1994], Eliminationsverfahren zur linearen und quadratischen Optimierung, Master's thesis, Universität Passau, FMI, Universität Passau.
- KAPUR D. [1995a], An approach for solving systems of parametric polynomial equations, in Saraswat and V. Hentenryck, eds, 'Principles and Practice of Constraint Programming', MIT press.
- KAPUR D. [1995b], Comparison of various multivariate resultant formulations, in A. Levelt, ed., 'International Symposium on Symbolic and Algebraic Computation, ISSAC'95', ACM Press, pp. 187–194.
- KAPUR D. [1998], Automated geometric reasoning: Dixon resultants, Gröbner bases, and characteristic sets, in D. Wang, ed., 'Automated Deduction in Geometry', Vol. 1360 of *LNAI*, Springer.
- KAPUR D. AND LAKSHMAN Y. N. [1992], Elimination methods: An introduction, in B. Donald, D. Kapur and J. Mundy, eds, 'Symbolic and Numerical Computation for Artificial Intelligence', Academic Press.
- KAPUR D., MUSSER D. R. AND NIE X. [1994], 'An overview of the Tecton proof system', *Theoretical Computer Science* **133**, 307–339.

- KAPUR D. AND NIE X. [1994], Reasoning about numbers in Tecton, Technical report, Dept. of Computer Science, State Univ. of New York, Albany.
- KAPUR D., SAXENA T. AND YANG L. [1994], Algebraic and geometric reasoning using Dixon resultants, in 'International Symposium on Symbolic and Algebraic Computation, ISSAC'94', ACM Press, pp. 108–113.
- KARMARKAR N. [1984], A new polynomial-time algorithm for linear programming, in '16th ACM Symposium on the Theory of Computing, STOC'84, Washington D.C.', pp. 302 – 311.
- KASPER T. [1998], A unifying logical framework for integer linear programming and finite domain constraint programming, PhD thesis, Technische Fakultät, Univ. des Saarlandes.
- KÄUFL T. [1988], Reasoning about systems of linear inequalities, in '9th International Conference on Automated Deduction, CADE-9, Argonne, USA', Springer, LNCS 310, pp. 563–572.
- KHACHIYAN L. G. [1979], 'A polynomial algorithm in linear programming', *Soviet Math. Doklady* **20**, 191 – 194.
- KIRCHNER C., KIRCHNER H. AND RUSINOWITCH M. [1990], 'Deduction with symbolic constraints', *Revue d'Intelligence Artificielle* **4**(3), 9–52.
- KIRCHNER H. [1995], On the use of constraints in automated deduction, in 'Constraint Programming: Basics and Trends', Springer, LNCS 910, pp. 128 – 146.
- KLEE V. AND MINTY G. J. [1972], How good is the simplex algorithm?, in O. Shisha, ed., 'Inequalities III', Academic Press, pp. 159 – 175.
- KOUBARAKIS M. [1996], Tractable disjunctions of linear constraints, in 'Principles and Practice of Constraint Programming, CP'96, Cambridge, MA', Springer, LNCS 1118, pp. 297 – 307.
- KREDEL H. [1993], MAS modula-2 algebra system, interactive usage, Technical report, Universität Passau. Available by anonymous ftp from [alice.fmi.uni-passau.de](ftp://alice.fmi.uni-passau.de).
- KREDEL H. AND WEISPFENNING V. [1988], 'Computing dimension and independent sets for polynomial ideals', *Journal of Symbolic Computation* **6**, 232–247.
- LAGARIAS J. C. [1985], 'The computational complexity of simultaneous diophantine approximation problems', *SIAM J. Computing* **14**(1), 196 – 209.
- LAMBERT J.-L. [1987], 'Une borne pour les générateurs des solutions entières positives d'une équation diophantienne linéaire', *C. R. Acad. Sci., Paris, Ser. I* **305**, 39 – 40.
- LANKFORD D. [1989], 'Non-negative integer basis algorithms for linear equations with integer coefficients', *Journal of Automated Reasoning* **5**, 25 – 35.
- LASSEZ J.-L. [1990], Querying constraints, in 'Proc. ACM Symp. on Principles of Database Systems', ACM-Press, pp. 288–298.
- LASSEZ J.-L. AND MAHER M. [1992], 'On Fourier's algorithm for linear arithmetic constraints', *Journal of Automated Reasoning* **9**, 373 – 379.
- LASSEZ J.-L. AND MCALOON K. [1992], 'A canonical form for generalized linear constraints', *Journal of Symbolic Computation* **13**, 1–24.
- LAZARD D. [1991], 'A new method for solving algebraic systems of positive dimension', *Discrete Applied Mathematics* **33**, 147–160.
- LAZARD D. [1992], 'Solving zero-dimensional algebraic systems', *Journal of Symbolic Computation* **13**, 117–131.
- LE CHENADEC P. [1986], *Canonical forms in finitely presented algebras*, Research Notes in TCS, Pitman.
- LENGAUER C. [1993], Loop parallelization in the polytope model, in 'CONCUR'93', Vol. 715 of LNCS, Springer, pp. 398–416.
- LENSTRA A. K., LENSTRA H. W. AND LOVÁSZ L. [1982], 'Factoring polynomials with rational coefficients', *Math. Annalen* **261**, 515 – 534.
- LENSTRA H. W. [1983], 'Integer programming with a fixed number of variables', *Mathematics of Operations Research* **8**(4), 538 – 548.
- LIPPOLD F. [1993], Implementierung eines Verfahrens zum Zählen reeller Nullstellen multivariater Polynome, Diploma thesis, Universität Passau, D-94030 Passau, Germany.

- LIPSHITZ L. [1978], 'The diophantine problem for addition and divisibility', *Trans. AMS* **235**, 271–283.
- LOMBARDI H. [1991], Effective real Nullstellensatz and variants, in M. T. and C. Traverso, eds, 'Effective Methods in Algebraic Geometry, MEGA'90', Birkhauser, pp. 263–288.
- LOOS R. AND WEISPFENNING V. [1993], 'Applying linear quantifier elimination', *The Computer Journal* **36**(5), 450–462. Special issue on computational quantifier elimination.
- LOVÁSZ L. AND SCARF H. E. [1992], 'The generalized basis reduction algorithm', *Mathematics of Operations Research* **17**(3), 751 – 764.
- MACKWORTH A. K. [1977], 'Consistency in networks of relations', *Artificial Intelligence* **8**, 99 – 118.
- MACMAHON P. A. [1916], *Combinatory Analysis*, Vol. II, The University Press, Cambridge.
- MADLENER K. AND REINERT B. [1998], String rewriting and Gröbner bases - a general approach to monoid and group rings, in M. Bronstein, J. Grabmeier and V. Weispfenning, eds, 'Symbolic Rewriting Techniques', Birkhäuser, Basel.
- MAKANIN G. [1979], 'Systems of standard equations in words in an n-layer alphabet of unknowns', *Sib. Math. J.* **19**, 448 – 454.
- MANDERS K. L. AND ADLEMAN L. [1978], 'NP-complete decision problems for binary quadratics', *Journal of Computer and System Sciences* **16**, 168 – 184.
- MARRIOTT K. AND STUCKEY P. J. [1998], *Programming with constraints*, MIT Press.
- MATIYASEVICH Y. V. [1970], 'Enumerable sets are Diophantine', *Soviet Math. Doklady* **11**(2), 354–358.
- MATIYASEVICH Y. V. [1993], *Hilbert's Tenth Problem*, MIT Press.
- MAZUR B. [1994], 'Questions of decidability and undecidability in number theory', *Journal of Symbolic Logic* **59**(2), 353 – 371.
- MEGIDDO N. [1984], 'Linear programming in linear time when the dimension is fixed', *Journal of the Association for Computing Machinery* **31**, 182–196.
- MELENK H., MÖLLER H. AND NEUN W. [1989], 'Symbolic solution of large stationary chemical kinetics problems', *IMPACT Comput. Sci. Eng.* **1**, 138–167.
- MIGNOTTE M. [1992], *Mathematics for computer algebra*, Springer.
- MISHRA B. [1993], *Algorithmic Algebra*, Springer Verlag.
- MOTZKIN T. S. [1936], Beiträge zur Theorie der linearen Ungleichungen, Doctoral dissertation, Universität Zürich.
- NELSON G. AND OPPEN D. C. [1979], 'Simplification by cooperating decision procedures', *ACM Transactions on Programming Languages and Systems* **1**(2), 245–257.
- NEMHAUSER G. L. AND WOLSEY L. A. [1988], *Integer and Combinatorial Optimization*, John Wiley.
- NIUWENHUIS R. AND RUBIO A. [2001], Paramodulation-based theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 7, pp. 371–443.
- OPPEN D. [1973], Elementary bounds for Presburger arithmetic, in 'Proceedings 5th ACM Symposium on Theory of Computing', ACM, pp. 34–37.
- PAN V. [1991], Complexity of algorithms for linear systems of equations, in E. Spedicato, ed., 'Computer algorithms for solving linear algebraic equations: the state of the art', Vol. F 77 of *NATO ASI series*, Springer.
- PASECHNIK D. V. [1998], On computing Hilbert bases via the Elliott-MacMahon algorithm. To appear in *Theoretical Computer Science*.
- PEDERSEN P., ROY M.-F. AND SZPIRGLAS A. [1993], Counting real zeroes in the multivariate case, in F. Eyssette and A. Galigo, eds, 'Computational Algebraic Geometry', Vol. 109 of *Progress in Mathematics*, Birkhäuser, Boston, Basel; Berlin, pp. 203–224. Proceedings of the MEGA 92.
- PERESSINI A., SULLIVAN F. AND UHL J. [1988], *The Mathematics of Nonlinear Programming*, Undergraduate Texts in Mathematics, Springer.

- PESANT G. [1995], Une approche géométrique aux contraintes arithmétiques quadratiques en programmation logique avec contraintes, PhD thesis, Université de Montréal.
- PESANT G. AND BOYER M. [1994], QUAD-CLP(R): Adding the power of quadratic constraints, in 'Principles and Practice of Constraint Programming', Springer, LNCS 874, pp. 95–108.
- PESCH M. [1994], Computing Comprehensive Gröbner Bases using MAS. User Manual.
- PHEIDAS T. [1994], 'Extensions of Hilbert's tenth problem', *Journal of Symbolic Logic* 59(2), 372 – 397.
- PLOTKIN G. [1972], Building-in equational theories, in B. Meltzer and D. Michie, eds, 'Machine Intelligence 7', Edinburgh Univ. Press, pp. 73–90.
- PORKOLAB L. AND KHACHIYAN L. [1997], Computing integral points in convex semi-algebraic sets, in 'Foundations of Computer Science, FOCS'97, Miami Beach', pp. 162–171.
- POTTIER L. [1991a], Minimal solutions of linear diophantine systems: bounds and algorithms, in 'Rewriting Techniques and Applications, RTA'91', Springer, LNCS 488.
- POTTIER L. [1991b], Sub-groups of \mathbb{Z}^n , standard basis, and linear diophantine systems, Technical Report RR-1520, INRIA, Sophia-Antipolis.
- PRESBURGER M. [1929], Über die Vollständigkeit eines gewissen Systems der Arithmetik, in 'Comptes rendues du 1er Congres des Mathematiques des Pays Slaves', Vol. 395, Warsaw, pp. 92–101.
- PUDLÁK P. [1997], 'Lower bounds for resolution and cutting plane proofs and monotone computations', *Journal of Symbolic Logic* 62(3), 981–988.
- PUDLÁK P. [1999], On the complexity of the propositional calculus, in 'Sets and Proofs, Invited papers from Logic Colloquium'97', Cambridge Univ. Press, pp. 197 – 218.
- REDDY C. AND LOVELAND D. [1978], Presburger arithmetic with bounded quantifier alternation, in 'Symposium on the Theory of Computing, STOC'78', ACM, pp. 320–325.
- RENEGAR J. [1991], Recent progress on the complexity of the decision problem for the reals, in 'Discrete and computational geometry, Proc. DIMACS Spec. Year Workshops 1989-90', Vol. 6 of *DIMACS, Ser. Discret. Math. Theor. Comput. Sci.*, AMS, pp. 287–308.
- RENEGAR J. [1992a], 'On the computational complexity and geometry of the first-order theory of the reals. Part I: Introduction. Preliminaries. The geometry of semi-algebraic sets. The decision problem for the existential theory of the reals', *Journal of Symbolic Computation* 13(3), 255–299.
- RENEGAR J. [1992b], 'On the computational complexity and geometry of the first-order theory of the reals. Part II: The general decision problem. Preliminaries for quantifier elimination', *Journal of Symbolic Computation* 13(3), 301–327.
- RENEGAR J. [1992c], 'On the computational complexity and geometry of the first-order theory of the reals. Part III: Quantifier elimination', *Journal of Symbolic Computation* 13(3), 329–352.
- ROMEUF J.-F. [1990], 'A polynomial algorithm for solving systems of two linear diophantine equations', *Theoretical Computer Science* 74(3), 329–340.
- ROUILLIER F. [1996], Algorithmes efficaces pour l'étude des zéros réels des systèmes polynomiaux, PhD thesis, Université Rennes.
- ROY M.-F. [1996], Basic algorithms in real algebraic geometry: from Sturm theorem to the existential theory of real numbers, in 'Lectures on Real Geometry in memoriam of Mario Raimondo', Vol. 23 of *Expositions in Mathematics*, de Gruyter, pp. 1–67.
- SARASWAT V. A. [1993], *Concurrent constraint programming*, MIT Press.
- SCARPELLINI B. [1984], 'Complexity of subcases of Presburger arithmetic', *Trans AMS* pp. 203–218.
- SCHRIJVER A. [1980], 'On cutting planes', *Annals of Discrete Mathematics* 9, 291 – 296.
- SCHRIJVER A. [1986], *Theory of Linear and Integer Programming*, John Wiley.
- SEBŐ A. [1990], Hilbert bases, Carathéodory's theorem and combinatorial optimization, in R. Kannan and W. R. Pulleyblank, eds, 'Integer Programming and Combinatorial Optimization, IPCO'90, Waterloo, ON, Canada', University of Waterloo Press.

- SEIDENBERG A. [1956a], 'An elimination theory for differential algebra', *Univ. California Publ. Math (N.S.)* **3**, 31–66.
- SEIDENBERG A. [1956b], 'Some remarks on Hilbert's Nullstellensatz', *Arch. Math.* **7**, 235–240.
- SHERALI H. D. AND SHETTY C. M. [1980], *Optimization with disjunctive constraints*, Vol. 181 of *Lecture Notes in Economics and Mathematical Systems*, Springer.
- SHOSTAK R. E. [1977], 'On the sup-inf method for proving Presburger formulas', *Journal of the Association for Computing Machinery* **24**(4), 529–543.
- SHOSTAK R. E. [1979], 'An efficient decision procedure for arithmetic with function symbols', *Journal of the Association for Computing Machinery* **26**(2), 351–360.
- SHOSTAK R. E. [1984], 'Deciding combinations of theories', *Journal of the Association for Computing Machinery* **31**(1), 1 – 12.
- SIEGEL C.-L. [1972], 'Zur Theorie der quadratischen Formen', *Nachr. Akad. Wiss. Goettingen, II. math.-phys. Kl.* pp. 21–46.
- SIEKMANN J. AND SZABÓ P. [1989], 'The undecidability of the D_A -unification problem', *Journal of Symbolic Logic* **54**(2), 402–414.
- SIT W. [1991], A theory for parametric linear systems, in S. Watt, ed., 'International Symposium on Symbolic and Algebraic Computation, ISSAC'91', ACM Press, pp. 112–121.
- SMART N. P. [1998], *The Algorithmic Resolution of Diophantine Equations - A Computational Cookbook*, Cambridge Univ. Press.
- SMITH H. J. S. [1861], 'On systems of linear indeterminate equations and congruences', *Phil. Trans. Royal Soc. London (A)* **151**, 293 – 326.
- SONTAG E. D. [1985], 'Real addition and the polynomial hierarchy', *Information Processing Letters* **20**, 115 – 120.
- SRIVASTAVA D. [1993], 'Subsumption and indexing in constraint query languages with linear arithmetic constraints', *Annals of Mathematics and Artificial Intelligence* **8**, 315 – 343.
- STANLEY R. P. [1973], 'Linear homogeneous diophantine equations and magic labelings of graphs', *Duke Math. J.* **40**, 607 – 632.
- STANLEY R. P. [1986], *Enumerative combinatorics*, Vol. I, The Wadsworth & Brooks/Cole Mathematics Series.
- STICKEL M. E. [1985], 'Automated deduction by theory resolution', *Journal of Automated Reasoning* **1**, 333–356.
- STORJOHANN A. [1997], Nearly optimal algorithms for computing Smith normal forms of integer matrices, in 'International Symposium on Symbolic and Algebraic Computation, ISSAC'96, Zurich', ACM Press, pp. 267 – 274.
- STORJOHANN A. AND LABAHN G. [1996], Asymptotically fast computation of Hermite normal forms of integer matrices, in 'International Symposium on Symbolic and Algebraic Computation, ISSAC'96, Zurich', ACM Press, pp. 259 – 266.
- STRASSEN V. [1969], 'Gaussian elimination is not optimal', *Numerische Mathematik* **13**, 354–356.
- STUCKEY P. J. [1991], 'Incremental linear constraint solving and detection of implicit equalities', *ORSA J. Comput.* **3**(4), 269 – 274.
- STURM T. [1997], Reasoning over networks by symbolic methods, Technical Report MIP-9719, FMI, Universität Passau, D-94030 Passau, Germany.
- STURM T. AND WEISPFENNING V. [1997], Rounding and blending of solids by a real elimination method, in A. Sydow, ed., 'Proceedings of the 15th IMACS World Congress on Scientific Computation, Modelling, and Applied Mathematics (IMACS 97)', Vol. 2, IMACS, Wissenschaft & Technik Verlag, Berlin, pp. 727–732.
- STURM T. AND WEISPFENNING V. [1998], Computational geometry problems in Redlog, in D. Wang, ed., 'Automated Deduction in Geometry', Vol. 1360 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin Heidelberg, pp. 58–86.
- STURMFELS B. [1993], *Algorithms in invariant theory*, Springer.
- SUZUKI N. AND JEFFERSON D. [1980], 'Verification decidability of Presburger array programs', *J. ACM* **27**, 191–205.

- TARDOS E. [1986], 'A strongly polynomial algorithm to solve combinatorial linear programs', *Operations Research* **34**, 250 – 256.
- TARSKI A. [1948], A decision method for elementary algebra and geometry, Technical report, University of California. 2nd ed., 1951.
- TAYLOR R. AND WILES A. [1995], 'Ring-theoretic properties of certain Hecke algebras', *Ann. Math., II. Ser.* **141**(3), 553–572.
- THOMAS R. [1995], 'A geometric Buchberger algorithm for integer programming', *Mathematics of Operations Research* **20**(4), 864 – 884.
- THOMAS R. [1998], Gröbner bases in integer programming, in D.-Z. Du and P. M. Pardalos, eds, 'Handbook of Combinatorial Optimization', Vol. 1, Kluwer, pp. 533–572.
- TOMÁS A. P. [1997], On solving linear diophantine constraints, PhD thesis, Univ. Porto.
- TOMÁS A. P. AND FILGUEIRAS M. [1997], Solving linear diophantine equations using the geometric structure of the solution space, in 'Rewriting Techniques and Applications, RTA-97, Sitges, Spain', Springer, LNCS 1232, pp. 269 – 283.
- TSANG E. [1993], *Foundations of Constraint Satisfaction*, Academic Press.
- TUNG S. P. [1987], 'Computational complexities of diophantine equations with parameters', *Journal of Algorithms* **8**, 324–336.
- TZANAKIS N. AND WEGER B. M. M. D. [1989], 'On the practical solution of the Thue equation', *J. Number Theory* **31**(2), 99–132.
- VAN DER WAERDEN B. L. [1940], *Moderne Algebra II*, 2nd edn, Springer, Berlin.
- VAN EMDE BOAS P. [1981], Another NP-complete partition problem and the complexity of computing short vectors in a lattice, Technical Report Report 81-04, Math. Inst., Univ. Amsterdam.
- VAN HENTENRYCK P. [1989], *Constraint satisfaction in logic programming*, MIT Press.
- VAN HENTENRYCK P. AND GRAF T. [1992], 'Standard forms for rational linear arithmetic in constraint logic programming', *Annals of Mathematics and Artificial Intelligence* **5**, 303–320.
- VAN HENTENRYCK P. AND SARASWAT V. [1996], 'Strategic directions in constraint programming', *ACM Computing Surveys* **28**(4), 701 – 726.
- VANDENBERGHE L. AND BOYD S. [1996], 'Semidefinite programming', *SIAM Review* **38**, 49 – 95.
- VANDERBEI R. J. [1997], *Linear programming : Foundations and extensions*, Kluwer.
- VAVASIS S. A. [1990], 'Quadratic programming is in NP', *Information Processing Letters* **36**(2), 73 – 77.
- VOLGER H. [1983], 'Turing machines with linear alternation, theories of bounded concatenation and the decision problem of first order theories', *Theoretical Computer Science* **23**, 333–337.
- VON ZUR GATHEN J. AND GERHARD J. [1999], *Modern computer algebra*, Cambridge University Press.
- WALLACE M. [1996], 'Practical applications of constraint programming', *Constraints* **1**, 139 – 168.
- WANG D. [1993a], An elimination method based on Seidenberg's theory and its applications, in F. Eyssette and A. Galligo, eds, 'Computational Algebraic Geometry (MEGA '92)', Birkhäuser Verlag, pp. 301–328.
- WANG D. [1993b], 'An elimination method for polynomial systems', *Journal of Symbolic Computation* **16**(2), 83–114.
- WANG D. [1995], Reasoning about geometric problems using an elimination method, in J. Pfalzgraf, ed., 'Automatic Practical Reasoning', Springer-Verlag, Wien, pp. 147–185.
- WAYNE K. D. [1999], A polynomial combinatorial algorithm for generalized minimum cost flow, in '31th ACM Symposium on the Theory of Computing, STOC'99, Atlanta.', pp. 11 – 18.
- WEISPFENNING V. [1975], 'Two model theoretic proofs of Rückert's Nullstellensatz', *Transactions AMS* **203**, 331–342.
- WEISPFENNING V. [1988], 'The complexity of linear problems in fields', *Journal of Symbolic Computation* **5**(1–2), 3–27.

- WEISPFENNING V. [1990a], 'The complexity of almost linear diophantine problems', *Journal of Symbolic Computation* 10(5), 395–403.
- WEISPFENNING V. [1990b], 'Existential equivalence of ordered abelian groups with parameters', *Archive for Mathematical Logic* 29, 237–248.
- WEISPFENNING V. [1992], 'Comprehensive Gröbner bases', *Journal of Symbolic Computation* 14, 1–29.
- WEISPFENNING V. [1994], Parametric linear and quadratic optimization by elimination, Technical Report MIP-9404, FMI, Universität Passau, D-94030 Passau, Germany.
- WEISPFENNING V. [1997a], Complexity and uniformity of elimination in Presburger arithmetic, in W. W. Kuchlin, ed., 'International Symposium on Symbolic and Algebraic Computation, ISSAC'97, Maui, Hawaii', ACM Press, New York, pp. 48–53.
- WEISPFENNING V. [1997b], 'Quantifier elimination for real algebra—the quadratic case and beyond', *Applicable Algebra in Engineering Communication and Computing* 8(2), 85–101.
- WEISPFENNING V. [1997c], 'Simulation and optimization by quantifier elimination', *Journal of Symbolic Computation* 24(2), 189–208. Special issue on applications of quantifier elimination.
- WEISPFENNING V. [1998], A new approach to quantifier elimination for real algebra, in B. Caviness and J. Johnson, eds, 'Quantifier Elimination and Cylindrical Algebraic Decomposition', Springer, Wien, New York, pp. 376–392.
- WEISPFENNING V. [1999], Mixed real-integer linear quantifier elimination, in S. Dooley, ed., 'International Symposium on Symbolic and Algebraic Computation, ISSAC'99', ACM Press, pp. 129–136.
- WEISPFENNING V. [2000], Deciding linear-transcendental problems, in V. Ganzha, E. Mayr and E. Vorozhtshov, eds, 'Computer Algebra in Scientific Computation - CASC 2000', Springer, pp. 423–438.
- WILES A. [1995], 'Modular elliptic curves and Fermat's last theorem', *Ann. Math., II. Ser.* 141(3), 443–551.
- WILLIAMS H. [1986], 'Fourier's method of linear programming and its dual', *American Mathematical Monthly* 93, 681–695.
- WOLSEY L. [1998], *Integer programming*, Wiley.
- YE Y. [1997], *Interior point algorithms : theory and analysis*, Wiley.
- ZHARKOV A. Y. AND BLINKOV Y. A. [1993], Involution approach to solving systems of algebraic equations, in 'Proc. IMACS'93', pp. 11–16.
- ZIEGLER G. M. [1999], Gröbner bases and integer programming, in A. M. Cohen, H. Cuypers and H. Sterk, eds, 'Some Tapas of Computer Algebra', Springer, pp. 168 – 183.

Index

A

affine	
combination	755
hull	755
subspace	755, 759
affine scaling algorithm	768
affinely independent	755
algebraically closed	757
algorithm	
affine scaling	768
cutting plane	785
Euclidean	779, 807
univariate polynomials	803
generalized basis reduction	784
interior point	768
lift-and-project	788
LLL	781
path following	769
primal-dual	769
potential reduction	769
rectangles	795
Simplex	764
slopes	795
arithmetic model	756
ascending set	808
atomic formula	757

B

barrier	
function	769
optimization problem	769
basic	
feasible	
index set	764
solution	763
solution	763
basis	
Gröbner	810
Hilbert	794
involution	812
lattice	781
Lovász-Scarf reduced	784
of a matrix	763
reduced Gröbner	811
binomial	793
Bland's rule	765
bound occurrence	757
bound reasoning	790
bounded quantifier	801
branch-and-bound	791

branch-and-cut	792
branch-and-infer	788
branch-and-relax	791
Buchberger reduction	809

C

canonical form	
generalized linear constraints	774
Cauchy bounds	805
center	
of an ellipsoid	766
central path	770
characteristic set	807
extended	808
Chvátal rank	787
clausal inequality	782
clause	
extended	799
prime extended	800
propositional	782
closed formula	757
combination	
affine	755
conic	755
convex	755
linear	755
complex and real quantifier elimination	816
conic	
combination	755
hull	755
subspace	755
constrained optimization	791
constraint	
branching	790
generalized linear	774
lower bounding	791
negative linear	774
non-primitive	788
numerical	754
positive linear	774
primitive	788
constraint programming	788
constraint satisfaction problem	790
constraint store	789
convex	
combination	755
hull	755
subspace	755
coprime	761
cutting plane	785, 792

algorithm	785
disjunctive	788
Gomory-Chvátal	786
proof	786
cylindrical algebraic decomposition ...	817

D

decision procedure	758
dependent	
affinely	755
linearly	755
dimension	
ideal	806
subspace	756
diophantine equation	819
superelliptic	821
Thue	821
diophantine set	820
disjunctive	
cutting plane	788
normal form	757
optimization	787
domain	
reasoning	790
reduction	789
dual	
linear program	763
duality theorem	763, 772

E

elementary	
integer column operations	780
elementary closure	787
elimination	
by substitution of test points	772
Fourier-Motzkin	771
Gauss-Jordan	761
Gaussian	759
ellipsoid	766
method	766
encoding length	
arithmetic model	756
Turing machine model	756
equation	
diophantine	819
linear	758
linear diophantine	779, 782, 794
polynomial	805
equivalent formula	757
Euclidean algorithm	779, 807
univariate polynomials	803
existential formula	757
extended clause	799

extreme ray	763
-------------------	-----

F

face	762
facet	762
Farkas Lemma	762, 772
fast matrix multiplication	760
feasible solution	763
first-order	
language	757
theory	758
formula	757
atomic	757
closed	757
equivalent	757
existential	757
prenex	757
quantifier-free	757
universal	757
Fourier-Motzkin elimination	771
free occurrence	757
full	
column rank	756
row rank	756
full-dimensional	756

G

Gauss-Jordan elimination	761
Gaussian elimination	759
generalized	
basis reduction	784
linear constraints	774
canonical form	774
resolution	799
generic system	815
Gomory-Chvátal cutting plane	786
Gröbner basis	807, 810
comprehensive	813, 817
reduced	811
Gröbner system	814

H

half space	762
height of Hilbert basis	798
Hermite normal form	780
Hilbert basis	794
height	798
Hilbert's Nullstellensatz	806
Hilbert's Tenth Problem	820
homogeneous polynomial	757
hull	
affine	755
conic	755
convex	755

integer	787
linear	755
hyperbolic programming	818
hyperplane	759

I

ideal	
toric	793
ideal membership problem	806
independence	
of negative constraints	774
inequality	
clausal	782
linear	761
valid	762
integer	
hull	787
linear optimization	785
linear programming	785, 788
interior point algorithm	768
isolating intervals	805

K

Karush-Kuhn-Tucker conditions	770
-------------------------------------	-----

L

lattice	781
basis	781
reduced	781
lift-and-project method	788
linear	
combination	755
diophantine equation .. 779, 782, 794	
equation	758
function	754
hull	755
inequality	761
optimization	763
program	
dual	763
primal	763
programming	763
parametric	775
relaxation	785
quantifier elimination	776
subspace	755
theory of real numbers	776
linearly independent	755
LLL algorithm	781
local consistency	789
Lovász-Scarf reduced basis	784

M

matrix	
--------	--

fast multiplication	760
positive definite	766
minimal	
solutions	794
minimum basis problem	782
monomial	793
multivariate polynomial ring	756

N

nearest lattice vector problem	781
negative linear constraint	774
non-decomposable solution	794
non-primitive constraint	788
normal form	
disjunctive	757
Hermite	780
Smith	781
Nullstellensatz	805
Hilbert	806
numerical	
constraint	754
domain	754
function	754

O

occurrence	
bound	757
free	757
optimal solution	763
optimization	791
disjunctive	787
integer linear	785
linear	763
problem	
barrier	769
semidefinite	819
order	
term	809

P

parameter	757
parametric	
linear programming	775
strongly	776
weakly	776
partial cylindrical algebraic	
decomposition	817
path	
central	770
path following algorithm	769
pigeon-hole problem	798
pivot element	760
pivoting rules	765
pointed	762

polyhedron	762
pointed	762
polynomial	754
binomial	793
constrained	814
equation	805
homogeneous	757
monomial	793
S-	810
polynomial ring	
multivariate	756
univariate	756
polytope	762
positive definite matrix	766
positive linear constraint	774
potential reduction algorithm	769
precise negative constraints	775
prefix	757
prenex formula	757
Presburger arithmetic	800
uniform	801
primal	
linear program	763
primal-dual path following algorithm	769
prime extended clause	800
primitive constraint	788
problem	
constrained optimization	791
constraint satisfaction	790
programming	
constraint	788
hyperbolic	818
integer linear	785, 788
linear	763
propagation	789
propositional clauses	782

Q

quantifier	
bounded	801
quantifier elimination	758
complex and real	816
linear	776
quantifier-free formula	757

R

Rabinovich trick	806
rank	756
ray	763
extreme	763
real-closed	757
reasoning	
bound	790

domain	790
rectangles algorithm	795
reduced lattice base	781
reduction	
Buchberger	809
Ritt	807
relaxation	789
linear programming	785
tight	792
remainder	803
resolution	798
generalized	799
resultant	815
u-	815
Dixon	816
mixed sparse	816
sparse	815
Sylvester	815
system	815
Ritt reduction	807

S

S-polynomial	810
semidefinite optimization	819
sensitivity analysis	775
set	
ascending	808
characteristic	807
diophantine	820
shortest lattice vector problem	781
Simplex algorithm	764
simplifying resolvents	800
slack variable	764
sliding objective function	772
slopes algorithm	795
Smith normal form	781
solution	
basic feasible	763
feasible	763
minimal	794
non-decomposable	794
optimal	763
square-free part	803
standard form	
linear optimization problem	764
strongly	
parametric	776
polynomial	756
Sturm-Sylvester	
sequence	803
theorem	
multivariate	812
subspace	

affine	755, 759
conic	755
convex	755
linear	755
superelliptic diophantine equation	821
Sylvester resultant	815

T

term	757
term order	809
degree-lex	809
degree-reverse-lex	809
inverse lex	809
lexicographic	809
theory	
first-order	758
linear of real numbers	776
Thue equation	821
tight relaxation	792
toric ideal	793

U

Uniform Presburger Arithmetic	801
univariate polynomial ring	756
universal formula	757

V

valid inequality	762
variety	757, 802
vertex	762
virtual substitution of test points	774
volume	
of an ellipsoid	766

W

weakly parametric	776
-------------------------	-----

Z

zero	756
non-trivial	757

The Automation of Proof by Mathematical Induction

Alan Bundy

SECOND READERS: David McAllester and Christoph Walter.

Contents

1	Introduction	847
1.1	Explicit vs Implicit Induction	848
1.2	Conventions	848
2	Induction Rules	848
2.1	Noetherian Induction	849
2.2	Constructor vs Destructor Style Induction Rules	849
2.3	Additional Universal Variables	850
3	Recursive Definitions and Datatypes	851
3.1	Recursive Datatypes	851
3.2	Recursive Definitions	852
3.3	Recursion/Induction Duality	854
3.4	The Need for Induction	854
4	Inductive Proof Techniques	855
4.1	Rewriting	855
4.2	Fertilization	858
4.3	Destructor Elimination	860
4.4	Termination of Rewriting	862
4.5	Decision Procedures	862
5	Theoretical Limitations of Inductive Inference	863
5.1	The Incompleteness of Inductive Inference	863
5.2	The Failure of Cut Elimination	864
6	Special Search Control Problems	865
6.1	Constructing an Induction Rule	865
6.2	Introducing an Intermediate Lemma	869
6.3	Generalising Induction Formulae	871
7	Rippling	876
7.1	Rippling Out	876
7.2	Simplification of Wave-Fronts	877
7.3	Rippling Sideways and In	878
7.4	The Advantages of Rippling	879
7.5	Selective Rewriting	879
7.6	Bi-Directional Rewriting	880

HANDBOOK OF AUTOMATED REASONING

Edited by Alan Robinson and Andrei Voronkov

© 2001 Elsevier Science Publishers B.V. All rights reserved

7.7	The Definition of Wave Annotation	882
7.8	Termination of Rippling	884
7.9	Automatic Annotation	885
7.10	Ripple Analysis	887
8	The Productive Use of Failure	890
8.1	Example: Speculating a Lemma	890
8.2	Example: Introducing a Sink	892
9	Existential Theorems	894
9.1	Synthesis Problems	894
9.2	Representing Existential Theorems	894
9.3	Extracting Recursive Definitions	895
9.4	Problems with Recursion Analysis	897
10	Interactive Theorem Proving	898
10.1	Division of Labour	898
10.2	Tactic-Based Provers	899
10.3	User Interfaces	899
11	Inductive Theorem Provers	900
11.1	The Boyer/Moore Theorem Prover	900
11.2	RRL	901
11.3	INKA	902
11.4	<i>Oyster/CIAM</i>	902
12	Conclusion	903
	Bibliography	904
	Main Index	909
	Name Index	911

1. Introduction

Inductive inference is theorem proving using induction rules. It is required for reasoning about objects, events or procedures containing repetition. As well as mathematical objects, like the natural numbers, these include: recursive data-structures, like lists or trees; computer programs containing recursion or iteration; and electronic circuits with feedback loops or parameterised components. Many properties of such objects cannot be proved without the use of induction (see Section 3.4, page 854). Inductive inference is thus a vital ingredient of formal methods for synthesis-ing, verifying and transforming software and hardware.

Induction rules infer universal statements incrementally. The premises of an induction consist of one or more base cases and one or more step cases. In a base case the conclusion of the rule is proved for a particular value; in a step case the conclusion is proved for a later value under the assumption that it is true for one or more previous values. The classic example of an induction rule is Peano induction:

$$\frac{P(0), \quad \forall n:\text{nat}. (P(n) \rightarrow P(s(n)))}{\forall n:\text{nat}. P(n)} \quad (1.1)$$

where $x:\tau$ means x is of type τ , nat is the type of natural numbers and $s(n) = n+1$. s is the successor function for natural numbers. This induction rule has one base case and one step case. In the base case the conclusion is proved for the value 0. In the step case the conclusion is proved for $s(n)$ under the assumption that it is true for n . $P(n)$ is called the *induction hypothesis*, $P(s(n))$ is called the *induction conclusion*, n is called the *induction variable* and $s(n)$ is called the *induction term*.

Unfortunately, the word "induction" is ambiguous in English. To avoid any misunderstanding we contrast mathematical induction with inductive learning. Inductive learning¹ is a rule of conjecture which takes the form:

$$\frac{P(c_0), \quad P(c_1), \quad P(c_2), \quad \dots, \quad P(c_m)}{\forall n:\text{nat}. P(n)}$$

i.e. if $P(n)$ can be proved for a sufficiently large number of particular cases then it is assumed true in general. It is a rule of *conjecture* rather than a rule of inference. In this chapter we will *not* be concerned with inductive learning.

Inductive inference requires special study because of negative theoretical results which do not apply to first-order theorem proving (see Section 5, page 863). These cause it to suffer additional search control problems. For instance, it is sometimes necessary to choose an induction rule, generalise the conjecture or to discover and prove an intermediate lemma. Any of these can introduce infinite branching points into the search space. New kinds of heuristic control are needed to deal with these special search problems.

¹Also called *philosophical induction*.

1.1. *Explicit vs Implicit Induction*

There have been two major approaches to the automation of inductive proof: explicit and implicit. This chapter is concerned with explicit induction, in which induction rules are explicitly incorporated into proofs.

In implicit induction the conjecture to be proved is added to the axioms. A Knuth-Bendix completion procedure is then applied to the whole system. If no inconsistency is derived by the procedure, then the conjecture is an inductive theorem. This method is also called *inductionless induction* or *inductive completion*. More details can be found in the chapter "Inductionless induction" by Hubert Comon in this book.

1.2. *Conventions*

In this chapter we will use the following conventions. The double shafted arrow, \Rightarrow , will be used to indicate the directed equality used in rewriting. The single shafted arrow, \rightarrow , will be used to represent logical implication.

Most research into inductive theorem proving has been restricted to the, so called, *quantifier-free* fragment of first-order logic. This means that all variables are free and, hence, implicitly universally quantified. The discussion below will be restricted to this fragment of logic, except in Section 8, page 890, where we will consider existentially quantified second-order variables, and Section 9, page 894, where we will consider existentially quantified first-order variables. Also, conjectures and induction rules will usually be presented in fully quantified form so that the types of the variables can be emphasised. Note that in quantifier-free form universal variables become free variables² in axioms and hypotheses, but become arbitrary constants³ in goals. We will follow the Prolog convention of starting all free variables with an upper case letter. Bound variables and constants will start with lower case letters.

Most of the example proofs discussed below will use backwards reasoning, from the original conjecture to derive \top , the truth value "true". So rules of inference, like rewriting (see §4.1) and induction (see §2), will be applied backwards. The current goal will be matched to the conclusion of the rule of inference and the premises of the rule will become the new goals.

2. Induction Rules

Peano induction is merely the simplest and best known inductive rule of inference. Similar structural induction rules are available for every kind of recursively defined data-structure, *e.g.* integers, lists, trees, sets, *etc.* Moreover, it is not necessary to

²Also called meta-variables. The translation of universal variables into free variables is affected by skolemisation.

³Also called skolem constants. This translation is affected by skolemising their negations and then re-negating. This is also called dual skolemisation.

traverse such data-structures in the obvious, stepwise manner; they can be traversed using any well-ordering. An extreme example occurs in a standard proof that the arithmetic mean is greater than or equal to the geometric mean. This uses an induction rule that traverses the natural numbers by first going up in multiples of 2 and then filling in the gaps by coming in down in steps of 1. Nor is induction restricted just to data-structures; it is possible to induce over the control flow of a computer program or the time steps of a digital circuit.

2.1. Noetherian Induction

All of these forms of induction are subsumed by a single, general schema of Noetherian induction⁴:

$$\frac{\forall x:\tau. (\forall y:\tau. y \prec x \rightarrow P(y)) \rightarrow P(x)}{\forall x:\tau. P(x)} \quad (2.1)$$

where \prec is some well-founded relation on the type τ , i.e. \prec is an irreflexive, anti-symmetric relation and there are no infinite, descending chains, like $\dots \prec a_n \prec \dots \prec a_3 \prec a_2 \prec a_1$. The data-structure, control flow, time step, etc., over which induction is to be applied, is represented by the type τ . The inductive proof is formalised in a many-sorted or many-typed logical system.

Success in proving a conjecture, P , by induction is highly dependent on the choice of x and \prec . There is an infinite variety of possible types, τ , and for most of these types, an infinite variety of possible well-orderings, \prec . Thus choosing an appropriate induction rule to prove a conjecture also introduces an infinite branching point into the search space. Controlling it, therefore, requires special heuristic techniques.

2.2. Constructor vs Destructor Style Induction Rules

Most inductive theorem proving systems construct customised induction rules for each conjecture rather than use the general well-founded induction rule directly. Such customised induction rules fall into two broad camps: constructor-style and destructor-style. In constructor-style rules the step cases have the form:

$$P(x_1) \wedge \dots \wedge P(x_m) \rightarrow P(c(x_1, \dots, x_m))$$

where $\forall i. x_i \prec c(x_1, \dots, x_m)$. Peano induction is an example of a constructor-style rule. In destructor-style rules the step cases have the form:

$$P(d_1(x)) \wedge \dots \wedge P(d_m(x)) \rightarrow P(x)$$

where $\forall i. d_i(x) \prec x$. In destructor-style, Peano induction would take the form:

$$\frac{P(0), \quad \forall n:\text{nat}. (n > 0 \wedge P(p(n)) \rightarrow P(n))}{\forall n:\text{nat}. P(n)}$$

⁴Also known as well-founded induction.

where p is the predecessor function for natural numbers, *i.e.*.

$$p(n) = \begin{cases} 0 & \text{if } n = 0 \\ m & \text{if } n = s(m) \end{cases}$$

In this chapter we will usually give constructor-style induction rules, recursive definitions and, hence, proofs. This is because most inductive proving techniques are more naturally described in constructor-style. In fact, when conjectures are stated in destructor-style it is usual to convert the resulting proof attempt to constructor-style at an early stage (see Section 4.3, page 860, for instance).

There are destructor-style induction rules which have no direct counterpart in constructor-style, for instance:

$$\frac{P(\text{nil}), \quad \forall x:\text{list}(\text{nat}). \forall n:\text{nat}. (n \in x \wedge P(\text{delete}(n, x)) \rightarrow P(n))}{\forall x:\text{list}(\text{nat}). P(x)}$$

but this can be converted into:

$$\frac{P(\text{nil}), \quad \forall x:\text{list}(\text{nat}). \forall n:\text{nat}. (P(x) \rightarrow P(n :: x))}{\forall x:\text{list}(\text{nat}). P(x)}$$

2.3. Additional Universal Variables

If an induction formula contains more than one universally quantified variable then there is a choice of induction variable. It is interesting to see what becomes of the universal variables which are *not* chosen as an induction variable. Consider, for instance, the induction formula $\forall n:\text{nat}. \forall m:\text{nat}. Q(n, m)$. Suppose we choose n as the induction variable. We can then apply the Peano induction rule (1.1) backwards with $\forall m:\text{nat}. Q(n, m)$ as $P(n)$. The step case of this induction is:

$$\forall n:\text{nat}. [(\forall m:\text{nat}. Q(n, m)) \rightarrow (\forall m:\text{nat}. Q(s(n), m))]$$

Note that the scope of the quantification of n is the whole step case, but the scopes of the two quantifications of m is restricted to the induction hypothesis and induction conclusion, respectively.

It is standard to strip the quantifiers from step cases and replace the implication with a turnstile. In this format the step case is:

$$Q(n, M) \vdash Q(s(n), m)$$

Note that the induction variable, n , becomes an arbitrary constant in both induction hypothesis and induction conclusion. The other universal variable, m , becomes an arbitrary constant, m , in the induction conclusion but a free variable in the induction hypothesis⁵. This means that when using the induction hypothesis to

⁵These translations are the effect of dual skolemisation of the step case. Note that the $\forall m$ in the induction hypothesis is in a position of negative polarity, so dual skolemisation turns this m into a free variable.

help prove the induction conclusion (see Section 4.2, page 858) we are not bound to match M to m . We can match M to any term, including one properly containing m , if desired. It is sometimes valuable to exploit this flexibility (see, for instance, Section 6.2.2, page 870).

3. Recursive Definitions and Datatypes

Recursion is frequently used in mathematics and programming both in the construction of classes of objects and in the definition of functions and programs. We call the former *recursive datatypes* and the latter *recursive definitions*. Induction is needed to reason about both of these.

3.1. Recursive Datatypes

Recursive datatypes are constructed by providing a set of *constructor functions* and then defining the datatype as the set of terms formed from them. If syntactically distinct terms are unequal then the datatype is called *free*, otherwise it is *non-free*. We discuss the free datatypes first.

3.1.1. Free Recursive Datatypes

We have already met one recursive datatype: the natural numbers. These are defined with the successor function s and the constant 0 as the constructor functions. For instance, the natural numbers are the set of terms: $\{0, s(0), s(s(0)), s(s(s(0))), \dots\}$, which we have abbreviated as *nat*. Note that we have been using the binary function $:$ to represent type membership, *i.e.* $n:nat$ says that n is a natural number.

Another recursive datatype we will meet frequently below is lists. Lists are a parameterised datatype, *i.e.* lists are of elements of some underlying type, *e.g.* natural numbers or letters. The constructors for lists are the empty list, *nil*, and the infix binary function $::$. The function $::$ takes an element of the underlying type and a list and returns a new list with the new element on the front of the old list. So the lists of type τ have the form: $\{nil, \alpha_1 :: nil, \alpha_2 :: \alpha_1 :: nil, \alpha_3 :: \alpha_2 :: \alpha_1 :: nil, \dots\}$, where the α_i are elements of type τ . We will abbreviate this as *list*(τ), *i.e.* the type of lists of natural numbers is *list*(*nat*). Lisp-style S-expressions (abbreviated as *sexpr*) differ from lists in permitting nesting of lists to any level. This datatype can be defined with the constructors *nil* and *cons*, where *cons* differs from $::$ by being able to take an S-expression as its first argument as well as its second. Similarly, we can construct one type of binary trees (abbreviated as *tree*(τ)) from the unary function *leaf* on labels and the binary function *node* on two trees.

The recursive datatypes of natural numbers, lists, S-expressions and trees are examples of free datatypes because terms are only equal if they are syntactically identical, *e.g.* $s(s(0)) \neq s(0)$.

3.1.2. Non-Free Recursive Datatypes

However, it is sometimes necessary to use *non-free* datatypes, i.e. datatypes in which syntactically different terms may be equal. A simple example is the integers defined with the constructors 0 , succ and pred , where the first two are like 0 and s for the natural numbers, but pred is the predecessor function for integers⁶, i.e. $\text{pred}(n) = n - 1$. The predecessor function is needed to define the negative integers: $\{0, \text{pred}(0), \text{pred}(\text{pred}(0)), \text{pred}(\text{pred}(\text{pred}(0))), \dots\}$. Unfortunately, this representation is redundant, since for instance $\text{succ}(\text{pred}(n)) = \text{pred}(\text{succ}(n)) = n$ for all n .

Another example of a non-free datatype is the sets. We can define $\text{set}(\tau)$, sets of elements of type τ , with the constructors empty and insert , analogous to nil and $::$ for lists. But this is not a free datatype because we have, for instance, the equalities:

$$\begin{aligned}\text{insert}(\alpha, \text{insert}(\alpha, \text{set})) &= \text{insert}(\alpha, \text{set}) \\ \text{insert}(\alpha, \text{insert}(\beta, \text{set})) &= \text{insert}(\beta, \text{insert}(\alpha, \text{set}))\end{aligned}$$

between non-identical terms.

3.2. Recursive Definitions

Functions are said to be defined recursively when the body of the definition refers to the function itself. We usually demand that such recursive definitions are *terminating*, i.e. that given some particular inputs the function will call itself only a finite number of times before stopping with some output. See Section 4.4, page 862 and the chapter “Rewriting” by Nachum Dershowitz in this book for more discussion of termination.

3.2.1. Structural Recursion

A common form of recursion is based on recursive datatypes and is called *structural* recursion. In its simplest form there is one equation for each constructor function of the datatype, e.g. the function $+$ can be defined on datatype *nat* as:

$$0 + Y = Y \tag{3.1}$$

$$s(X) + Y = s(X + Y) \tag{3.2}$$

Note that the recursive call of $+$ on the RHS of the second equation has as its first argument, X , which is the argument of the constructor s on the LHS. It is clear that structural recursions like this terminate since $+$ is called on a syntactically simpler first argument on the RHS than on the LHS. For free datatypes, like *nat*, it is

⁶Note that pred differs from p , the predecessor function for natural numbers, since $p(0) = 0$, whereas $\text{pred}(0) = -1$.

also clear that structural recursion is *well-defined*, i.e. $+$ is neither under- nor over-defined. It is not under-defined because there is an equation for each combination of inputs. It is not over-defined because there is only one equation for each combination of inputs.

3.2.2. Non-Free Datatypes and Over-Definition

This is not clear for non-free datatypes. There is a danger here of over-definition, i.e. of giving different values to calls with equal inputs. Consider, for instance, the definition of $+$ for integers.

$$\begin{aligned} 0 + Y &= Y \\ \text{succ}(X) + Y &= \text{succ}(X + Y) \\ \text{pred}(X) + Y &= \text{pred}(X + Y) \end{aligned}$$

Since $\text{succ}(\text{pred}(n)) = \text{pred}(\text{succ}(n)) = n$ we have to check the side-condition:

$$\text{succ}(\text{pred}(n)) + m = \text{pred}(\text{succ}(n)) + m = n + m$$

otherwise, the definition of $+$ could introduce a contradiction into the theory. In this case this side-condition is easily proved. However, if we had erroneously defined $+$ as:

$$\begin{aligned} 0 + Y &= Y \\ \text{succ}(X) + Y &= \text{succ}(X + Y) \\ \text{pred}(X) + Y &= 0 \end{aligned}$$

then we would find that:

$$\text{succ}(0) = \text{succ}(\text{pred}(0) + 0) = \text{succ}(\text{pred}(0)) + 0 = \text{pred}(\text{succ}(0)) + 0 = 0$$

i.e. that $+$ is now over-defined, enabling a proof of $\text{succ}(0) = 0$. So recursive definitions over non-free datatypes carry additional proof obligations to ensure that functions are not over-defined. For a discussion of some additional problems with non-free datatypes and one way to solve them see [Sengler 1996].

3.2.3. Non-Structural Recursions

Recursive definitions can take many other forms than constructor-style structural recursions. For instance, destructors can be used instead of constructors. Consider, for instance, this alternative definition of $+$ on the natural numbers:

$$\begin{aligned} X + Y &= \text{if } X = 0 \text{ then } Y \\ &\quad \text{else } s(p(X) + Y) \end{aligned}$$

Sometimes the recursive calls of the algorithm are not simply on the arguments of the constructors. Consider, for instance, this definition of *quicksort*.

$$\text{quicksort}(\text{nil}) = \text{nil}$$

$$\text{quicksort}(H :: T) = \text{quicksort}(\text{lesseq}(H, T)) \text{ } <> \text{ } (H :: \text{quicksort}(\text{greater}(H, T)))$$

where the recursive calls are on terms containing the arguments of the constructor function. Termination of such definitions is non-trivial. We need to find a well-founded relation, \prec , such that $\text{lesseq}(H, T) \prec H :: T$ and $\text{greater}(H, T) \prec H :: T$. In this case \prec can be defined as:

$$K \prec L \leftrightarrow \text{length}(K) < \text{length}(L)$$

3.3. Recursion/Induction Duality

There is an intimate relationship between induction rules and recursive definitions. Not only is induction required for reasoning about recursively defined objects, but there is a duality between the forms of recursive definitions and the forms of induction rules. For instance, the two step recursion below that defines the *even* predicate:

$$\begin{aligned} \text{even}(0) &\leftrightarrow \top \\ \text{even}(s(0)) &\leftrightarrow \perp \\ \text{even}(s(s(N))) &\leftrightarrow \text{even}(N) \end{aligned} \tag{3.3}$$

(where \top is “true” and \perp is “false”) is structurally similar to the following two step induction rule:

$$\frac{P(0), \quad P(s(0)), \quad \forall n:\text{nat}. (P(n) \rightarrow P(s(s(n))))}{\forall n:\text{nat}. P(n)}$$

We will see in Section 6.1.1, page 866 that this duality between recursion and induction can be exploited when choosing an induction rule to prove properties of recursive functions. We can also construct new induction rules by analogy to recursive definitions. When proving that a recursively defined function terminates we must exhibit a well-founded relation that decreases when the function is applied. This relation can then be used to instantiate the Noetherian induction schema, (2.1).

We will see examples below of inductions and recursions based on more complex well-founded relations than the simple structural ones provided by recursive datatypes.

3.4. The Need for Induction

Inductive inference is an essential tool for reasoning about recursively defined datatypes and functions. Without it, many true formulae cannot be proved. Recursive and induction are opposite sides of the same coin. Recursion specifies the behaviour of a function over all members of a datatype; induction allows us to exploit the restriction of variables to that datatype.

For instance, consider the formula:

$$\forall x:\text{nat}. x + 0 = x \tag{3.4}$$

This is true for the natural numbers and is readily proved by induction from the recursive definition of $+$. Peano induction reduces it to two cases: the base case $0 + 0 = 0$ and the step case $x + 0 = x \vdash s(x) + 0 = x$. The base case is an instance of (3.1), the base equation of the definition of $+$; the step case is readily proved by applying (3.2), the step equation of the definition of $+$, and then the induction hypothesis.

However, without the use of induction (3.4) is not provable. To see this we need only exhibit a model of the recursive definition of $+$ in which (3.4) is false. To form this model we augment the natural numbers with the additional base element $0'$ to form the datatype nat' . Think of nat' as the disjoint union of 'red' naturals $(0, s(0), s(s(0)), \dots)$ and 'blue' naturals $(0', s(0'), s(s(0')), \dots)$. Let the true formulae in this model be just those formulae made true by the definition of $+$. So, in particular, $0' + 0 = 0'$ is false. Therefore,

$$\forall x: nat'. x + 0 = x \quad (3.5)$$

is false. But if (3.4) were provable solely from the recursive definition of $+$ then (3.5) would also be provable from them. Therefore, induction⁷ is needed to prove (3.4). Induction allows us to exploit the fact that x in (3.4) ranges over nat and not some larger datatype, like nat' .

4. Inductive Proof Techniques

Apart from the application of induction rules, a number of proof techniques are used in inductive proofs. These range from standard techniques, like rewriting, to more specialised techniques like *fertilization*, [Boyer and Moore 1988a][§10.5], where the induction hypothesis is used to prove the induction conclusion.

Many of these techniques are of use in non-inductive proofs as well as inductive proofs and some of these are discussed in more detail in other chapters of this book. In these cases a short account is included here for completeness and a pointer is given to the other chapters for more detail.

4.1. Rewriting

The definition of a function or predicate is often given as a set of recursion equations or equivalences⁸. Many of the lemmas required in proofs are also often equations. A common technique in inductive theorem proving is to express these equations as rewrite rules and apply them using the rewrite rule of inference backwards:

$$\frac{lhs \Rightarrow rhs, \quad P[rhs\phi]}{P[sub]}$$

⁷Or some principle of equivalent power.

⁸Note that equivalences can be regarded as equations over the booleans, so references to "equations" below will include equivalences.

where $P[sub]$ means sub is a sub-term of formula P , called the *redex*, $rhs\phi$ means ϕ is a substitution of terms for variables which is applied to rhs and $lhs\phi \equiv sub$. An example is:

$$\frac{2 \times X \Rightarrow X + X, \quad even(n + n)}{even(2 \times n)}$$

Sometimes we will want to use conditional rewrite rules. To apply these we will need following modified version of the rule of inference:

$$\frac{Cond \rightarrow lhs \Rightarrow rhs, \quad P[rhs\phi], \quad Cond\phi}{P[sub]}$$

where $Cond$ is the condition. Recall that we will usually be applying the rewriting rule of inference backwards.

For more details about rewriting see the chapter “Rewriting” by Nachum Der-showitz in this book.

4.1.1. Definitions and Lemmas as Rewrite Rule Sets

It is standard to turn recursive definitions of functions into sets of rewrite rules, oriented so that the defined term is replaced by its definition. Thus the definitions of infix addition, $+$, on *nat* and infix list append, $<>$, on *list*(τ) will be given as rewrite rules as follows:

$$\begin{aligned} 0 + Y &\Rightarrow Y \\ s(X) + Y &\Rightarrow s(X + Y) \end{aligned} \tag{4.1}$$

$$nil <> L \Rightarrow L \tag{4.2}$$

$$(H :: T) <> L \Rightarrow H :: (T <> L) \tag{4.3}$$

Functions can be defined recursively on one or more of their arguments. These are called their *recursive arguments*. The recursive arguments of $+$ and $<>$ are their first arguments.

Lemmas can also be presented as rewrite rules. The decision to represent them in this way constitutes a commitment to their direction of application. In some cases this is uncontroversial, for instance the commuted version of rule (4.1) is often useful as the rule:

$$X + s(Y) \Rightarrow s(X + Y) \tag{4.4}$$

But in other cases it is more problematic. For instance, both orientations of associative laws are frequently required.

$$\begin{aligned} X <> (Y <> Z) &\Rightarrow (X <> Y) <> Z \\ (X <> Y) <> Z &\Rightarrow X <> (Y <> Z) \end{aligned}$$

But if both are included their unrestricted use can cause non-termination of rewriting. Commutative laws cannot be included in either orientation without risking non-termination.

$$X + Y \Rightarrow Y + X$$

One solution to such problems is to build such problematic lemmas into the unification algorithm, so that they are not needed as rewrite rules. For more details on how this is done see the chapter on "Unification theory" by Franz Baader and Wayne Snyder in this book.

4.1.2. Implicational Rewrites

We can use rewrite rules based on implication as well as equations and equivalences. Care needs to be taken with such rules to ensure that their application is sound. In particular, the direction of their application depends on the polarity of the redex and also on the direction of reasoning. An example of a frequently used family of implications is the replacement axioms of equality:

$$X_1 = Y_1 \wedge \dots \wedge X_n = Y_n \rightarrow f(X_1, \dots, X_n) = f(Y_1, \dots, Y_n)$$

Where f is the constructor of a free datatype, *e.g.* s or $::$, these implications can be strengthened to equivalences:

$$\begin{aligned} X_1 &= Y_1 \leftrightarrow s(X_1) = s(Y_1) \\ X_1 &= Y_1 \wedge X_2 = Y_2 \leftrightarrow X_1 :: X_2 = Y_1 :: Y_2 \end{aligned} \quad (4.5)$$

but in general, they cannot, *e.g.*

$$\begin{aligned} (X_1 &= Y_1 \wedge X_2 = Y_2) \rightarrow (X_1 + X_2 = Y_1 + Y_2) \\ (X_1 &= Y_1 \wedge X_2 = Y_2) \rightarrow (X_1 <> X_2 = Y_1 <> Y_2) \end{aligned}$$

are one way only. Confusingly, the legal orientation of replacement axioms is often the reverse of their implication direction, *i.e.*

$$\begin{aligned} (X_1 + X_2 &= Y_1 + Y_2) \Rightarrow (X_1 = Y_1 \wedge X_2 = Y_2) \\ (X_1 <> X_2 &= Y_1 <> Y_2) \Rightarrow (X_1 = Y_1 \wedge X_2 = Y_2) \end{aligned}$$

This is because the usual use of these implicational rules is backwards and applied to positions of positive polarity.

4.1.3. Examples: Base and Step Cases

We will illustrate the use of rewriting with two examples of their use: in the base and step case of a simple inductive proof.

Consider the associativity of $<>$:

$$\forall x:\text{list}(\tau) \forall y:\text{list}(\tau) \forall z:\text{list}(\tau). x <> (y <> z) = (x <> y) <> z$$

We will choose a simple one-step list induction on x using the induction rule:

$$\frac{P(\text{nil}), \quad \forall h:\tau. \forall t:\text{list}(\tau). P(t) \rightarrow P(h :: t)}{\forall l:\text{list}(\tau). P(l)} \quad (4.6)$$

The base case of the proof is⁹:

$$\text{nil} <> (y <> z) = (\text{nil} <> y) <> z$$

This can be rewritten with two applications of (4.2) as follows:

$$\begin{aligned} \text{nil} <> (y <> z) &= (\text{nil} <> y) <> z \\ y <> z &= (\text{nil} <> y) <> z \\ y <> z &= y <> z \end{aligned}$$

In future, where two or more rewrites are independent, as here, we will save space by applying them in parallel¹⁰.

The step case of the proof is:

$$t <> (Y <> Z) = (t <> Y) <> Z \vdash (h :: t) <> (y <> z) = ((h :: t) <> y) <> z$$

This can be rewritten with three applications of (4.3), followed by an application of (4.5), the replacement rule for ::.

$$\begin{aligned} t <> (Y <> Z) &= (t <> Y) <> Z \vdash (h :: t) <> (y <> z) = ((h :: t) <> y) <> z \\ &\vdash h :: (t <> (y <> z)) = (h :: (t <> y)) <> z \\ &\vdash h :: (t <> (y <> z)) = h :: ((t <> y) <> z) \\ &\vdash h = h \wedge t <> (y <> z) = (t <> y) <> z \end{aligned}$$

The induction conclusion now contains an instance of the induction hypothesis and the proof can be simply completed (see Section 4.2, page 858). Note that Y and Z in the induction hypothesis are free variables, as explained in Section 2.3, page 850, but this extra flexibility was not required in this simple proof.

4.2. Fertilization

The purpose of rewriting in the step cases is to make the induction conclusion look more like the induction hypothesis. The hypothesis can then be used to help prove the conclusion. This can be clearly seen in the example step case in §4.1.3. Here, when rewriting terminated, an instance of the hypothesis was embedded in the conclusion.

⁹Recall that induction rules are applied backwards.

¹⁰Unfortunately, this is *not* something that most rewrite based provers can manage.

The next step is to use the induction hypothesis to prove the induction conclusion. After rewriting we have the situation: $IH \vdash IC[IH\phi]$ i.e. the induction conclusion, IC , contains an instance of the induction hypothesis, $IH\phi$, embedded within it. We can then use the rules of logic to rewrite this to: $IH \vdash IC[\top]$, i.e. we use the following rule of inference backwards:

$$\frac{IH \vdash IC[\top]}{IH \vdash IC[IH\phi]}$$

Following Boyer and Moore¹¹, we call this step *strong fertilization*: the hypothesis fertilizes the conclusion.

In the example in §4.1.3 we go from:

$$t \lt;> (Y \lt;> Z) = (t \lt;> Y) \lt;> Z \vdash h = h \wedge t \lt;> (y \lt;> z) = (t \lt;> y) \lt;> z$$

to:

$$t \lt;> (Y \lt;> Z) = (t \lt;> Y) \lt;> Z \vdash h = h \wedge \top$$

which rapidly simplifies to \top , completing the step case.

Sometimes, rewriting gets stuck before a *complete* instance of the hypothesis appears in the conclusion, but a large part of the hypothesis *does* appear in the conclusion. For instance, if the conjecture is an equation then one side of the conclusion may have an instance of the corresponding side of the hypothesis embedded in it. This will happen in our example if we do not have the replacement rule for $::$ available as a rewrite rule. The final stage of the rewriting process is then:

$$t \lt;> (Y \lt;> Z) = (t \lt;> Y) \lt;> Z \vdash h :: (t \lt;> (y \lt;> z)) = h :: ((t \lt;> y) \lt;> z)$$

An instance of each side of the hypothesis is embedded in each side of the conclusion. We can choose one side of the conclusion and replace the embedded side of the hypothesis with the other side of the hypothesis; effectively using the hypothesis as a rewrite rule. In our example this produces either:

$$t \lt;> (Y \lt;> Z) = (t \lt;> Y) \lt;> Z \vdash h :: (t \lt;> (y \lt;> z)) = h :: (t \lt;> (y \lt;> z))$$

or:

$$t \lt;> (Y \lt;> Z) = (t \lt;> Y) \lt;> Z \vdash h :: ((t \lt;> y) \lt;> z) = h :: ((t \lt;> y) \lt;> z)$$

depending on which side we choose to replace. In either case the remaining goal is now trivially proved. This is called *weak fertilization*. In general, weak fertilization leaves a more complex goal to prove than is the case with strong fertilization, but it can be applied in situations where strong fertilization cannot. The residue left after weak fertilization often requires a nested induction to prove, whereas strong

¹¹They called what we call weak fertilization, *cross fertilization*. We have dropped the "cross" and introduced the terms "weak" and "strong" to distinguish two different forms of fertilization.

fertilization usually completes the step case. So strong fertilization leads to shorter proofs and is to be preferred when available. The general form of weak fertilization is:

$$\frac{IH_1 = IH_2 \vdash IC_1[IH_2\phi] = IC_2}{IH_1 = IH_2 \vdash IC_1[IH_1\phi] = IC_2}$$

or

$$\frac{IH_1 = IH_2 \vdash IC_1 = IC_2[IH_1\phi]}{IH_1 = IH_2 \vdash IC_1 = IC_2[IH_2\phi]}$$

Note that these rules of inference can be further generalised to replace $=$ with any transitive relation with appropriate monotonicity properties, but we omit the details of this here.

4.3. Destructor Elimination

In this section we redeem the promise of Section 2.2, page 849 to show how destructor-style proofs can be converted to constructor-style ones.

The discussion of rewriting (Section 4.1, page 855) and fertilization (Section 4.2, page 858) above adopted an implicitly constructor induction stance. The induction term occurred in the induction conclusion; the rewriting was of the induction conclusion; and the fertilization matched the induction hypothesis to a sub-expression of the induction conclusion.

If a destructor style induction is used then the induction term appears in the induction hypothesis. It would be tempting to think that a dual process could then take place, with the hypothesis being rewritten and fertilization matching the conclusion to a sub-expression of the hypothesis. Unfortunately, the dual of fertilization is not true, *i.e.*

$$\frac{IH[\top] \vdash IC}{IH[IC\phi] \vdash IC}$$

is not a sound rule of inference, and nor are the duals of weak fertilization.

One solution to this problem is to try to turn destructor style step cases into constructor style ones, by replacing destructor functions in the hypothesis with constructor functions in the conclusion. This process is usually called *destructor elimination*, [Boyer and Moore 1988a][§10.4, p225]. Its application is not restricted to step cases, and we define it for any formula. Moreover, the concepts of “destructor function” and “constructor function” are interpreted loosely — they can be any functions the user so specifies.

Suppose that a formula contains occurrences of the expressions $d_i(x)$, where each d_i is a destructor function. Destructor elimination takes place in two steps:

1. A (possibly conditional) rewrite rule of the form:

$$Cond \rightarrow X \Rightarrow c(d_1(X), \dots, d_n(X)) \quad (4.7)$$

where c is a constructor function, is applied once to each occurrence of x not dominated by a d_i . Note that this may require a *Cond*/ \neg *Cond* case split if *Cond* is not already true.

2. All occurrences of x now occur within some d_i . Each $d_i(x)$ is generalised to a new variable y_i . See Section 6.3.2, page 873 for an explanation of this form of generalisation.

If a rewrite rule of form (4.7) is available then the application of this destructor elimination process will remove all occurrences of d_i in favour of c .

To see the effect of destructor elimination on a destructor-style inductive proof, consider the following schematic step case:

$$x \neq 0 \wedge \Phi(p(x)) \vdash \Phi(x) \quad (4.8)$$

where $x : nat$. For stage 1 of destructor elimination we appeal to the rewrite rule:

$$X \neq 0 \rightarrow X \Rightarrow s(p(X))$$

to rewrite (4.8) to:

$$s(p(x)) \neq 0 \wedge \Phi(p(x)) \vdash \Phi(s(p(x)))$$

Note that the condition of this rewrite rule is true by hypothesis. Stage 2 is to generalise all occurrences of $p(x)$ to y , giving:

$$s(y) \neq 0 \wedge \Phi(y) \vdash \Phi(s(y))$$

All occurrences of the destructor function, p , have now been replaced by the constructor function, s . This step case can be further simplified to:

$$\Phi(y) \vdash \Phi(s(y))$$

which is a constructor style step case.

Destructor elimination is not restricted to structural inductions, like the example above. It can also be used, for instance, to transform:

$$y \neq 0 \wedge \Phi(\text{remainder}(x, y)) \wedge \Phi(\text{quotient}(x, y)) \vdash \Phi(x)$$

to:

$$y \neq 0 \wedge \Phi(r) \wedge \Phi(q) \vdash \Phi(q \times y + r)$$

exchanging the destructor functions, *remainder* and *quotient* for the constructor functions $+$ and \times . Stage 1 of this destructor elimination uses the conditional rewrite rule:

$$Y \neq 0 \rightarrow X \Rightarrow \text{quotient}(X, Y) \times Y + \text{remainder}(X, Y)$$

In future we will usually assume that destructor elimination has been or could be applied and draw most of our examples from constructor style inductive proofs.

4.4. Termination of Rewriting

A common proof technique is to apply a set of rewrite rules to a goal until no further rules apply. The rewritten goal is then said to be in normal form. It is highly desirable if this rewriting process terminates. This question is equivalent to the halting problem (the problem of proving that computer programs terminate) so is undecidable. A partial solution has been provided by a collection of techniques which, although necessarily incomplete, have a high success rate when applied to the rewrite rule sets that arise in practical theorem proving. Each of these techniques involve defining a measure from terms to a well-founded set, *e.g.* the natural numbers, and showing that this measure decreases strictly each time a rewrite is applied. Since the measure is well-founded it cannot decrease indefinitely, *e.g.* it must eventually reach 0. More details about termination techniques can be found in the chapter “Rewriting” by Nachum Dershowitz in this book.

A particular case of this problem of especial interest is the termination of the rewrite rules which define a function. The proof of termination of these rules is usually a condition of accepting the definition as well-formed. The termination measures developed for this purpose are often recycled as the well-founded measures of induction rules (see Section 6.1, page 865 for more details).

4.5. Decision Procedures

Many of the problems to be solved by an inductive theorem prover fall within a decidable class and can be solved by a decision procedure. This is especially true of many of the subproblems generated during the proof of an inductive theorem. So decision procedures are an important component of inductive provers. These include the following:

Tautology Checkers: Many subproblems can be generalised into formulae of propositional logic. This generalisation may require regarding non-propositional formulae as propositional variables. If these generalised formulae are tautologies then the subproblem is true. Ordered Binary Decision Diagrams (OBDDs) provide a basis for efficient tautology checking and were devised for use in hardware verification, [Bryant 1992].

Congruence Closure: The propagation of equalities is an important ingredient of efficient theorem proving, *i.e.* if two terms are known to be equal we need to use this fact to simplify the conjecture. Congruence closure does this by forming equivalence classes for all subterms in a conjecture and propagating results between them. In its simplest version the negation of conjecture is put in disjunctive normal form and equivalence classes are constructed for each disjunct, [Nelson and Oppen 1980]. Positive equalities are used to update the equivalence classes and negative equalities are tested against them to see if there is a contradiction.

Presburger Arithmetic Procedures: Presburger identified a decidable fragment of integer arithmetic, [Presburger 1930, Stansifer 1984]. It consists of

formulae about equalities and inequalities between terms involving addition, but not multiplication. The equivalent real number fragment is also decidable. The integer fragment is particularly important in software verification as conjectures in Presburger arithmetic often arise from proof obligations about iterative loops, for instance. Many decision procedures exist for these fragments and are in common use in inductive provers, where they are often called linear arithmetic procedures. [Boyer and Moore 1988*b*] is an interesting discussion of the integration of one of these procedures into an inductive prover.

Combination Procedures: Decision procedures for two disjoint decidable theories can be combined. [Nelson and Oppen 1979, Shostak 1984] describe two such combination mechanisms.

Decision procedures often have unattractive theoretical worst case complexity, *e.g.* super-exponential. This does not always make them unusable. They can have empirically acceptable average case complexity when applied to problems of practical interest. In any case, the theoretical complexity of the alternative, full-blown inductive theorem proving, is usually much worse.

It is important to use decision procedures flexibly. [Boyer and Moore 1988*b*] reports that very few subproblems in a standard corpus were exactly in the Presburger fragment, but many more were almost in it and could be solved by a decision procedure augmented with a few additional facts about the terms, *e.g.* that the minimum element of an array was not bigger than the maximum element. Boyer and Moore flexibly interfaced their decision procedure to the rest of their theorem prover so that each could call the other and, hence, provide these additional facts to the decision procedure. Time spent by the interface components was much greater than time spent in the theorem prover.

Decision procedures are described in more detail in [Clarke and Schlingloff 2001, Fermüller et al. 2001] (Chapters 24 and 25) in this book.

5. Theoretical Limitations of Inductive Inference

Some negative results from mathematical logic impose special restrictions on inductive inference. In particular, results of Gödel and Kreisel introduce infinite branching points into the search space and show that it is impossible to build a complete inductive theorem prover.

5.1. The Incompleteness of Inductive Inference

Gödel's first incompleteness theorem, [Gödel 1931, Heijenoort 1967], states that in any formal theory of arithmetic there will be formulae which are true but unprovable. This incompleteness theorem is true of any non-trivial inductive theory. It puts a limit on the power of any automated¹² inductive theorem prover.

¹²And any human one too.

One way to see this result is as a limitation of our ability to construct the induction rule(s) required to prove each conjecture. We have already seen in Section 2, page 848 that there are an infinite number of different induction rules (or an infinite number of ways of instantiating Noetherian induction). In Section 6.1, page 865 we will investigate mechanisms for tailoring induction rules to the current conjecture. Gödel's incompleteness theorem tells us that, however sophisticated our induction rule construction mechanism, there will always be true formulae whose proof requires an induction rule that it cannot construct.

This limitation is illustrated in [Kirby and Paris 1982]. The theory of natural numbers can be formalised using Peano induction, (1.1). More complex induction rules can be derived from Peano induction. However, Kirby and Paris show that the termination of a simple recursive function (Goodstein's function) cannot be shown using any of these induction rules, but can be shown using the ϵ_0 induction rule. This induction rule is based on a complex well-founded relation which cannot be derived from Peano induction. Of course, we could add the ϵ_0 induction rule to our theory of natural numbers, but Gödel's incompleteness theorem tells us there would then be further true formulae, whose proof required even more complex forms of induction, and which were unprovable even within our extended theory.

This limitation is also related to the undecidability of the halting problem [Turing 1936-7]. Turing showed that there was no algorithm which could determine whether an arbitrary relation was well-founded. So we cannot construct all valid induction rules by instantiating the Noetherian induction scheme with all possible relations and then rejecting those that are not well-founded. Turing's result shows that this programme will not work, since there is no algorithm for deciding which of these potential induction rules is valid.

5.2. The Failure of Cut Elimination

Gentzen's original formalisation of sequent calculus contained the cut rule:

$$\frac{A, \Gamma \vdash \Delta, \quad \Gamma \vdash A}{\Gamma \vdash \Delta}$$

The cut rule allows us to first prove Δ with the aid of A and then eliminate A by proving it from Γ . A is called the *cut formula*.

If the cut rule is used backwards by a theorem prover then it introduces infinite branching into the search space; the cut formula can be *any* formula. The problem cannot be avoided by only using the cut rule forwards. Then we will be forced to use other sequent calculus rules forwards too. Several of these have formulae in the conclusion that do not occur in the premises, so will also cause infinite branching.

Gentzen recognised this problem and partially solved it by proving the cut elimination theorem, [Gentzen 1969]. He showed that the cut rule was redundant for first-order theories¹³. Unfortunately, Kreisel has shown that Gentzen's cut elimi-

¹³One source of confusion in this discussion is that the cut rule is similar to resolution. Of course, resolution is used in a forwards direction, so it does not cause infinite branching.

nation does not hold for inductive theories, [Kreisel 1965]. The cut rule must be retained and is a source of infinite branching.

The problem of infinite branching cannot be avoided by using an alternate formalisation of logic, *e.g.* natural deduction, resolution, etc; it recurs, in a different guise, in every formalism. It is possible to reorganise some of the infinite branching points so that they occur as an infinite series of finite branching points, but this does not significantly improve the size of the search space. Nor is this just a theoretical problem with little practical import. As we will see, the cut rule is needed even for many quite simple theorems.

6. Special Search Control Problems

Inductive inference can be automated by adding one or more induction rules to an automatic theorem prover. Unfortunately, this is not enough. An unbounded number of induction rules are required¹⁴. The cut rule is also needed. As we have seen, these requirements introduce infinite branching points into the search space. Thus inductive inference suffers from search control problems that do not arise in non-inductive, first-order, automatic theorem proving. Specialised heuristics have been developed for dealing with these search problems.

The cut rule is frequently required for two tasks: generalising the induction formula; and introducing an intermediate lemma. The cut formula is the generalised formula or the lemma. We, therefore, require heuristics for deciding when a generalisation or lemma are required and for determining their form.

Below we discuss further the search control problems of: induction rule choice; lemma introduction; and generalisation.

6.1. Constructing an Induction Rule

The success of an inductive proof attempt depends critically on the choice of induction rule. A good choice will lead to a short proof. For instance, a few rewritings of the induction conclusion will lead to fertilization and a successful conclusion. A bad choice may require multiple nested inductions or cause the proof to become stuck altogether.

Since there are an infinite number of possible induction rules it is not possible to prestore them; they must be constructed dynamically according to need. Heuristics are used to construct an induction rule that has a good chance of success on the current conjecture. The standard heuristic is called *recursion analysis*¹⁵. It uses the definitions of recursive functions appearing in the conjecture.

¹⁴Or the ability to construct new well-founded relations for Noetherian induction.

¹⁵Walther, [Walther 1994a], calls it *the* induction heuristic, but we will see that there are alternative heuristics.

6.1.1. Recursion Analysis

The starting point is to identify occurrences of recursively defined functions in the conjecture whose recursion arguments contain universally quantified variables. These variables are candidate induction variables. The recursive definition of each function suggests a dual induction rule. The idea underlying recursion analysis is that using an induction rule based on recursive definitions will facilitate the use of these recursive definitions in the base and step case proofs.

For instance, consider the conjecture:

$$\forall x:\text{nat}.\forall y:\text{nat}.\text{even}(x) \wedge \text{even}(y) \rightarrow \text{even}(x + y) \quad (6.1)$$

even is a recursively defined function and the occurrence of *even*(*x*) has a universally quantified variable, *x*, in its recursion argument. From the recursive definition of *even*, (3.3), we can construct the induction rule:

$$\frac{P(0), \quad P(s(0)), \quad \forall x:\text{nat}.\ (P(x) \rightarrow P(s(x)))}{\forall x:\text{nat}.\ P(x)} \quad (6.2)$$

The occurrence of *even*(*y*) suggests the same induction rule, but with *y* as the induction variable. The occurrence of *even*(*x* + *y*) does not suggest an induction rule, because its recursion argument does not contain a variable. However, + is also recursively defined and the occurrence of *x* + *y* has a variable, *x*, in its recursion argument, which suggests the induction rule:

$$\frac{P(0), \quad \forall x:\text{nat}.\ (P(x) \rightarrow P(s(x)))}{\forall x:\text{nat}.\ P(x)} \quad (6.3)$$

Note that + is defined on its first argument (see (4.1)), so that *y* is not a recursion argument of + and, therefore, does not suggest an induction rule.

We now see how the right choice of induction rule facilitates the subsequent use of recursive definitions. For instance, if the conjecture contains an occurrence of *x* + *y* and we apply induction rule (6.3) then the induction conclusion will contain the term *s*(*x*) + *y*. The step case of the recursive definition of + can then be applied to this term. On the other hand, if we erroneously choose *y* as the induction variable then the step case will contain the term *x* + *s*(*y*), and the recursive definition does not apply to this term. So if we used one step induction on *y* this occurrence of *s*(*y*) would be difficult to move and would prevent strong fertilization. Similar remarks apply to the base case.

The above process produces a variety of suggestions for induction rules. Some of these can be rejected as inferior to others and the rest can be combined together to produce a final induction rule. In our example the choice of *x* as induction variable is superior to *y*. This is because each occurrence of *x* in (6.1) is in a recursion argument position, so each occurrence of *x* in the induction conclusion can be potentially be rewritten by a recursive definition, making an eventual fertilization more likely. These occurrences of *x* are said to be *unflawed*. Universal variables, like *x*, with only unflawed occurrences are said to be *unflawed* induction variable candidates. In

contrast, the second occurrence of y in (6.1) is not in a recursive argument position. This occurrence will be replaced by $s(s(y))$, say, and it will not be possible to rewrite this occurrence, preventing strong fertilization. This occurrence of y is said to be *flawed*. Universal variables, like y , with some flawed occurrences, are said to be *flawed* induction variable candidates.

6.1.2. Subsumption of Induction Rules

So x is the best choice for induction variable, but this leaves two possibilities for induction rule: (6.2) and (6.3). Fortunately, rule (6.2) *subsumes* rule (6.3), *i.e.* rule (6.2) can stand-in for rule (6.3). This means that rule (6.3) is inferior to rule (6.2) and can be rejected. Roughly speaking, induction rule A *subsumes* induction rule B iff each induction term of a step case of A consists of repeated forms of an induction term of a step case of B (see [Stevens 1988] for a more detailed discussion). In our example $s(s(x))$ is a repeated form of $s(x)$. Using induction rule (6.2) the induction conclusion is:

$$\forall y:\text{nat. } \text{even}(s(s(x))) \wedge \text{even}(y) \rightarrow \text{even}(s(s(x)) + y)$$

The expression $\text{even}(s(s(x)))$ can be rewritten to $\text{even}(x)$ using the recursive definition of *even*. The expression $\text{even}(s(s(x)) + y)$ can be rewritten first to $\text{even}(s(s(x) + y))$ and then to $\text{even}(s(s(x + y)))$ by the recursive definition of $+$ and then to $\text{even}(x + y)$ with the definition of *even*, *i.e.* induction rule (6.2) facilitates a double application of the recursive definition of $+$, instead of the single application we would have gotten from rule (6.3). Here we see consequences of using a subsuming rule instead of the originally suggested rule. The induction conclusion now matches the induction hypothesis and the step case is finished.

Note that the rule (6.3) does not work so well. The induction conclusion is:

$$\forall y:\text{nat. } \text{even}(s(x)) \wedge \text{even}(y) \rightarrow \text{even}(s(x) + y)$$

Now the expression $\text{even}(s(x))$ cannot be rewritten and the step case proof is stuck. Rule (6.3) applied to y would encounter the same problem, *i.e.* $\text{even}(s(y))$ cannot be rewritten. So a subsumed induction rule cannot stand in for a subsuming one.

6.1.3. Containment of Induction Rules

Another way in which one induction rule can be inferior to others is *containment*.

Induction rule A *contains* induction rule B iff each step case of B is contained in some step case of A. A step case

$$\text{Cond}^A \wedge IH_1^A \wedge \dots \wedge IH_m^A \rightarrow IC$$

contains a step case

$$\text{Cond}^B \wedge IH_1^B \wedge \dots \wedge IH_n^B \rightarrow IC$$

iff $\text{Cond}^B \rightarrow \text{Cond}^A$ and each IH_j^B is also one of the IH_i^A . Note that these conditions make any instantiation of rule A logically easier to prove than a corresponding instantiation of rule B. So rule A is preferred over rule B. [Walther 1994a]

provides a calculus for describing induction rules, called *r-descriptions*, and gives a *containment formula* for defining and proving containment which is based on *r-descriptions*. To illustrate containment, consider the following two induction rules for S-expressions:

$$\frac{P(\text{nil}), \quad \forall e:\text{sexpr}. e \neq \text{nil} \wedge P(\text{cdr}(e)) \rightarrow P(e)}{\forall e:\text{sexpr}. P(e)} \quad (6.4)$$

$$\frac{P(\text{nil}), \quad \forall e:\text{sexpr}. e \neq \text{nil} \wedge P(\text{car}(e)) \wedge P(\text{cdr}(e)) \rightarrow P(e)}{\forall e:\text{sexpr}. P(e)} \quad (6.5)$$

Note that the non-inductive conditions and induction conclusions of the step cases of the two rules are the same and the induction hypotheses of rule (6.4) are a subset of those of rule (6.5). Thus the step case of rule (6.5) contains that of rule (6.4), so rule (6.5) contains rule (6.4). If both of these rules were suggested by recursion analysis then rule (6.4) should be rejected as inferior.

Unfortunately, containment and subsumption can sometimes order induction rules in opposite orders. Containment orders induction rules in terms of logical implication, but subsumption is a more heuristically based order which orders according to how easily standard proof methods will apply. Where they conflict containment usually makes better suggestions. Subsumption can also be used to tie-break where containment fails to distinguish. Unfortunately, containment has only been defined for destructor-style induction rules, so subsumption is usually used for constructor-style rules.

6.1.4. Combining Induction Rules

Sometimes no rule is suggested which subsumes or contains all the others. Then it is necessary to generalise and combine the rule suggestions to construct a rule which *does* subsume or contain them all. For instance, suppose our conjecture is about S-expressions and recursion analysis yields the following two suggestions:

$$\frac{P(\text{nil}), \quad \forall e:\text{sexpr}. e \neq \text{nil} \wedge P(\text{car}(e)) \rightarrow P(e)}{\forall e:\text{sexpr}. P(e)}$$

$$\frac{P(\text{nil}), \quad \forall e:\text{sexpr}. e \neq \text{nil} \wedge P(\text{cdr}(e)) \rightarrow P(e)}{\forall e:\text{sexpr}. P(e)}$$

Neither of these contains the other. However, both are contained by the more general rule:

$$\frac{P(\text{nil}), \quad \forall e:\text{sexpr}. e \neq \text{nil} \wedge P(\text{car}(e)) \wedge P(\text{cdr}(e)) \rightarrow P(e)}{\forall e:\text{sexpr}. P(e)}$$

which can be constructed from the two initially suggested induction rules by combining them. In this case the combination consists of conjoining the induction hypotheses of the two original rules. [Walther 1994a] defines combination with respect to containment as the *separated union* of the *r-descriptions* of two induction rules.

Combination can also be defined with respect to subsumption. Walther also defines various ways to generalise induction rules. Note that rule combination does not necessarily preserve the well-foundedness of the induction relation, so this may need to be proved after a merge has been made.

Recursion analysis was invented by Boyer & Moore, [Boyer and Moore 1979]. It was further developed by Stevens, [Stevens 1988], and Walther, [Walther 1994a]. Together they have constructed a range of techniques for preferring, generalising and combining initial induction rule suggestions. These are often successful in producing customised induction rules which lead to successful and short proofs of the current conjecture. However, further research is required, *e.g.* to extend containment to constructor-style induction rules and to incorporate within it some of the successful features of subsumption.

6.2. Introducing an Intermediate Lemma

Sometimes a lemma required to complete the proof is not already available and is not deducible from the existing theory without a nested application of induction. This is a consequence of the failure of cut elimination for inductive theories (see Section 5.2, page 864). Such lemmata must be conjectured and then proved as sub-goals. In non-inductive theorem proving, conjecturing lemmata is non-essential, because any lemmas needed will be generated by inference with existing rules. However, if induction is required to prove a lemma then inference alone is not sufficient, and the lemma must be conjectured.

6.2.1. Example: Reverse-Reverse

As an example, consider the conjecture:

$$\forall l: \text{list}(\tau). \text{rev}(\text{rev}(l)) = l$$

where *rev* reverses a list and is defined by the following rewrite rules:

$$\begin{aligned} \text{rev}(\text{nil}) &\Rightarrow \text{nil} \\ \text{rev}(H :: T) &\Rightarrow \text{rev}(T) <> (H :: \text{nil}) \end{aligned} \quad (6.6)$$

Recursion analysis will suggest the one-step list induction rule (4.6) on *l*. The step case of this induction develops as follows:

$$\begin{aligned} \text{rev}(\text{rev}(t)) &= t \vdash \text{rev}(\text{rev}(h :: t)) = h :: t \\ &\vdash \text{rev}(\text{rev}(t) <> (h :: \text{nil})) = h :: t \end{aligned}$$

but then gets stuck; no rewrite rules apply. Nor will strong fertilization apply¹⁶.

One solution is to introduce a distributive lemma of *rev* over *<>*., namely:

$$\text{rev}(X <> Y) \Rightarrow \text{rev}(Y) <> \text{rev}(X) \quad (6.7)$$

¹⁶Although weak fertilization will — see below Section 6.3.2, page 873.

This allows the step case to continue:

$$\begin{aligned}
& rev(rev(t)) = t \vdash rev(rev(t) <> (h :: nil)) = h :: t \\
& \quad \vdash rev(h :: nil) <> rev(rev(t)) = h :: t \\
& \quad \vdash (rev(nil) <> (h :: nil)) <> rev(rev(t)) = h :: t \\
& \quad \vdash (nil <> (h :: nil)) <> rev(rev(t)) = h :: t \\
& \quad \vdash (h :: nil) <> rev(rev(t)) = h :: t \\
& \quad \vdash h :: (nil <> rev(rev(t))) = h :: t \\
& \quad \vdash h :: rev(rev(t)) = h :: t \\
& \quad \vdash h = h \wedge rev(rev(t)) = t
\end{aligned}$$

which contains the induction hypothesis. Fertilization leaves the trivial goal $h = h \wedge \top$.

This does not solve the search problem. We need a heuristic to suggest or construct lemma (6.7). We will provide such a heuristic in Section 8.1, page 890. Other heuristics are also available, for instance the proof schema learning technique of [Kolbe and Walther 1998].

6.2.2. Example: Generalised Rotate Length

As another example, consider the conjecture:

$$\forall l : list(\tau). \forall k : list(\tau). rotate(length(l), l <> k) = k <> l \quad (6.8)$$

where $rotate(n, l)$ removes the first n elements from list l and appends them to the end and $length$ measures the length of the list. This conjecture says that if we remove $length(l)$ elements from $l <> k$ and put them at the end then we form the list $k <> l$.

The functions $rotate$ and $length$ are defined by the following rewrite rules:

$$\begin{aligned}
& length(nil) \Rightarrow 0 \\
& length(H :: T) \Rightarrow s(length(T)) \\
& rotate(0, L) \Rightarrow L \\
& rotate(s(N), nil) \Rightarrow nil \\
& rotate(s(N), H :: T) \Rightarrow rotate(N, T <> (H :: nil))
\end{aligned}$$

Recursion analysis will suggest the one-step list induction rule (4.6) applied either on l or k . l has two unflawed and one flawed occurrences and k has one unflawed and one flawed occurrences. There is not much to choose between the two variables, but some heuristics would give l a slight edge, so we will choose it.

The step case of this induction develops as follows:

$$\begin{aligned}
 & \text{rotate}(\text{length}(t), t \text{ <> } K) = K \text{ <> } t \\
 & \vdash \text{rotate}(\text{length}(h :: t), (h :: t) \text{ <> } k) = k \text{ <> } (h :: t) \\
 & \vdash \text{rotate}(\text{length}(t), h :: (t \text{ <> } k)) = k \text{ <> } (h :: t) \\
 & \vdash \text{rotate}(\text{length}(t), (t \text{ <> } k) \text{ <> } (h :: \text{nil})) = k \text{ <> } (h :: t)
 \end{aligned}$$

At this point the proof is stuck: no rewrite rules apply and both weak and strong fertilization are inapplicable.

We need two new lemmas: one to unstick the LHS and one to unstick the RHS. These are:

$$\begin{aligned}
 & (X \text{ <> } Y) \text{ <> } Z \Rightarrow X \text{ <> } (Y \text{ <> } Z) \\
 & L \text{ <> } (H :: T) \Rightarrow (L \text{ <> } (H :: \text{nil})) \text{ <> } T
 \end{aligned}$$

The first lemma is the associativity of list append and the second can be thought of as a special case of associativity where the middle list is a singleton. Note that they are required with the orientation given, although the opposite orientation is equally natural. As in Section 6.2.1, page 869 the question arises as to what heuristic might suggest or construct these lemmas. Again we will return to this question in Section 8.1, page 890.

With these lemmas the step case of the proof can continue and is now successful:

$$\begin{aligned}
 & \text{rotate}(\text{length}(t), t \text{ <> } K) = K \text{ <> } t \\
 & \vdash \text{rotate}(\text{length}(t), (t \text{ <> } k) \text{ <> } (h :: \text{nil})) = k \text{ <> } (h :: t) \\
 & \vdash \text{rotate}(\text{length}(t), t \text{ <> } (k \text{ <> } (h :: \text{nil}))) = (k \text{ <> } (h :: \text{nil})) \text{ <> } t
 \end{aligned}$$

Strong fertilization now applies. Note that K is instantiated to $k \text{ <> } (h :: \text{nil})$.

As discussed in Section 2.3, page 850, additional universal variables in the conjecture become free variables in the induction hypothesis and arbitrary constants in the induction conclusion. These free variables can be instantiated to compound terms when matching hypothesis to conclusion. This gives us more flexibility in the step case of the proof; a flexibility which is exploited in this example.

6.3. Generalising Induction Formulae

Suppose we are trying to prove a conjecture, C . Generalisation consists of constructing a generalised conjecture, G , and both proving G and $G \rightarrow C$ ¹⁷.

Sometimes a conjecture cannot be proved without first being generalised. This is another consequence of the failure of cut elimination for inductive theories. The generalization must be strong enough that the induction hypothesis can be used to prove the induction conclusion, but not so strong that it is not a theorem. Various techniques for generalisation have been developed.

¹⁷For all the heuristic generalisation techniques we will consider the subgoal $G \rightarrow C$ is true by construction.

6.3.1. Example: Generalising Apart

The need for generalisation can arise in even quite simple conjectures. Consider the following special case of the associativity of $\langle \rangle$.

$$\forall l:\text{list}(\tau). l \langle \rangle (l \langle \rangle l) = (l \langle \rangle l) \langle \rangle l$$

Where the only axioms available are the equality axioms and those arising from recursive definitions, *e.g.* (4.3).

Recursion analysis will suggest the one-step induction rule (4.6) on l , even though l is flawed, because there is no alternative. Unfortunately, these flaws cause the proof to fail. Note that the 3rd, 5th and 6th occurrences of l are not in recursive argument positions. However, the induction rule will replace these occurrences with the induction term, $h :: t$. So the induction conclusion has the form:

$$(h :: t) \langle \rangle ((h :: t) \langle \rangle (h :: t)) = ((h :: t) \langle \rangle (h :: t)) \langle \rangle (h :: t)$$

The step case of the recursive definition of $\langle \rangle$, rewrite rule (4.3), is able to rewrite the 1st, 2nd and 4th occurrences of l , but not the other three occurrences. Moreover, the 2nd occurrence can only be rewritten once. The induction conclusion, therefore, gets stuck in the state:

$$h :: (t \langle \rangle (h :: (t \langle \rangle (h :: t)))) = h :: ((t \langle \rangle (h :: t)) \langle \rangle (h :: t))$$

to which neither weak nor strong fertilization applies, causing the proof attempt to fail if no generalisation is allowed.

To unstick the proof we must *generalise apart* the occurrences of l to give the conjecture:

$$\forall l:\text{list}(\tau). k:\text{list}(\tau). l \langle \rangle (k \langle \rangle k) = (l \langle \rangle k) \langle \rangle k$$

Recursion analysis will still suggest a one-step induction on l , but this time it is unflawed. The step case then proceeds as follows:

$$\begin{aligned} t \langle \rangle (k \langle \rangle k) &= (t \langle \rangle k) \langle \rangle k \vdash (h :: t) \langle \rangle (k \langle \rangle k) = ((h :: t) \langle \rangle k) \langle \rangle k \\ &\vdash h :: (t \langle \rangle (k \langle \rangle k)) = (h :: (t \langle \rangle k)) \langle \rangle k \\ &\vdash h :: (t \langle \rangle (k \langle \rangle k)) = h :: ((t \langle \rangle k) \langle \rangle k) \\ &\vdash h = h \wedge t \langle \rangle (k \langle \rangle k) = (t \langle \rangle k) \langle \rangle k \end{aligned}$$

to which strong fertilization applies, allowing the proof to be completed.

The generalisation worked by restricting the occurrences of the induction variable to unflawed ones. This removed from the induction conclusion those occurrences of the induction term which could not be rewritten. Note that the 2nd occurrence of l was replaced by k even though it was unflawed. To have left it as l would have caused two problems. Firstly, it would have resulted in a non-theorem:

$$\forall l:\text{list}(\tau), k:\text{list}(\tau) l \langle \rangle (l \langle \rangle k) = (l \langle \rangle k) \langle \rangle k$$

Secondly, the 2nd occurrence would have become stuck after the first rewrite. Deciding which occurrences of the induction variable to generalise apart is a non-trivial problem. It may be necessary to try several combinations before the correct one is found. No one has yet found a heuristic which always chooses the correct combination first time.

We also need a heuristic to decide to try generalising apart in the first place. Various heuristics have been proposed for this, all based on the analysis of initial failed proofs (see, for instance, [Hesketh 1991]).

6.3.2. Example: Generalising a Sub-Term

Consider again the *rev-rev* conjecture:

$$\forall l: \text{list}(\tau). \text{rev}(\text{rev}(l)) = l$$

from Section 6.2.1, page 869 and the point at which the step case gets stuck:

$$\text{rev}(\text{rev}(t)) = t \vdash \text{rev}(\text{rev}(t) <> (h :: \text{nil})) = h :: t$$

An alternative method of unsticking this step case is to use weak fertilization (see Section 4.2, page 858). The induction hypothesis is used as a rewrite rule right to left and applied to the RHS of the induction conclusion. This yields:

$$\text{rev}(\text{rev}(t) <> (h :: \text{nil})) = h :: \text{rev}(\text{rev}(t))$$

We can now try to solve this new goal, using induction if necessary. Unfortunately, the presence of nested *rev* functions will cause the step case again to get stuck. However, note that term *rev(t)* occurs on both sides of the equation. This can be generalised to a new variable, e.g. *k*, and the resulting formula:

$$\text{rev}(k <> (h :: \text{nil})) = h :: \text{rev}(k)$$

is still a theorem. Moreover, the problem of nested *revs* has now gone away. This generalised conjecture is much easier to prove. For instance, the step case is now:

$$\begin{aligned} \text{rev}(t' <> (h :: \text{nil})) &= h :: \text{rev}(t') \\ \vdash \text{rev}((h' :: t') <> (h :: \text{nil})) &= h :: \text{rev}(h' :: t') \\ \vdash \text{rev}(h' :: (t' <> (h :: \text{nil}))) &= h :: (\text{rev}(t') <> (h' :: \text{nil})) \\ \vdash \text{rev}(t' <> (h :: \text{nil})) <> (h' :: \text{nil}) &= (h :: \text{rev}(t')) <> (h' :: \text{nil}) \quad (6.9) \\ \vdash \text{rev}(t' <> (h :: \text{nil})) &= h :: \text{rev}(t') \wedge h' :: \text{nil} = h' :: \text{nil} \end{aligned}$$

to which strong fertilization applies, completing the proof. This generalisation worked by generalising away a subterm which caused difficulty during rewriting.

Note that in step (6.9) it was necessary to apply the rewrite rule (4.3) from right to left, i.e. in the wrong orientation. We will propose a solution to this problem in Section 7.6, page 880.

The most common heuristic for generalising subterms is to do so only when all occurrences of a variable, say x , occur in a common term, say $f(x)$. All occurrences of $f(x)$ (and hence x) are then replaced with a new variable y . Another heuristic is to restrict generalisation to variables in recursive argument positions. The new variable, y , will then be a candidate for an induction variable. The generalisation can sometimes make possible an induction and subsequent rewriting that were not previously available. Even with these heuristics, over-generalisation to a false conjecture can occur — a problem we will address in §6.3.4.

6.3.3. Example: Introducing New Universal Variables

Another kind of generalisation is illustrated by the rotate length conjecture:

$$\forall l : \text{list}(\tau). \text{rotate}(\text{length}(l), l) = l \quad (6.10)$$

which is a special case of conjecture (6.8). This conjecture says that if we remove $\text{length}(l)$ elements from l and put them at the end then we recover the original list l .

Recursion analysis will suggest the one-step list induction rule (4.6) on l . The step case of this induction develops as follows:

$$\begin{aligned} \text{rotate}(\text{length}(t), t) &= t \vdash \text{rotate}(\text{length}(h :: t), h :: t) = h :: t \\ &\vdash \text{rotate}(s(\text{length}(t)), h :: t) = h :: t \\ &\vdash \text{rotate}(\text{length}(t), t <> (h :: \text{nil})) = h :: t \end{aligned}$$

At this point the proof is stuck: no rewrite rule applies and strong fertilization fails. Weak fertilization succeeds, but the resulting conjecture is harder to prove than the original one.

One solution is to generalise the original conjecture by introducing an additional universally quantified variable. The generalised rotate length conjecture is:

$$\forall l : \text{list}(\tau). \forall k : \text{list}(\tau). \text{rotate}(\text{length}(l), l <> k) = k <> l$$

which is conjecture (6.8) proved in Section 6.2.2, page 870.

This generalisation succeeds because importing an additional universal variable into the conjecture enables us to exploit the extra flexibility described in Section 2.3, page 850. In Section 8.2, page 892 we describe a heuristic for suggesting and constructing this kind of generalisation.

Many other forms of generalisation are possible. Figure 1 lists some of these. More discussion can be found in [Hummel 1990].

6.3.4. The Problem of Over-Generalisation

A major problem with generalisation is the danger of over-generalisation, *i.e.* of generalising a theorem into a non-theorem. For instance, consider the theorem:

$$\forall l : \text{list}(\text{nat}). \text{sort}(\text{sort}(l)) = \text{sort}(l)$$

Original	Generalisation	Discussion
$A \rightarrow B$	$A \leftrightarrow B$	implication to equivalence
$A \rightarrow B$	B	dropping a condition
$A \vee B$	A	dropping a disjunct
A	$A \wedge B$	adding a conjunct
$f(s) = f(t)$	$s=t$	cancelling common structure

Figure 1: Some Other Forms of Generalisation

where *sort* is one of many functions for sorting lists of numbers into numerical order. An automated inductive prover might generalise this theorem into the non-theorem:

$$\forall k: \text{list}(\text{nat}). \text{sort}(k) = k \quad (6.11)$$

by replacing the term *sort(l)* by the new variable *k* using the generalisation technique outlined in Section 6.3.2, page 873.

One partial solution to this problem is to check any newly generalised formula with a counter-example finder, [Protzen 1992]. A simple counter-example finder might generate a small number of variable-free instances of the generalised formula and check that each evaluates to \top . For instance, if we checked (6.11) above with the list $[2, 1]$ for *k* then *sort(k)* = *k* would evaluate to \perp and the generalisation could be rejected. Simple checking of this kind works in the majority of cases because over-generalisations are rarely false in any subtle way.

Another partial solution is to try to modify the over-generalised non-theorem back into a theorem. For instance, non-theorem (6.11) can be modified to the theorem:

$$\forall k: \text{list}(\text{nat}). \text{ordered}(k) \rightarrow \text{sort}(k) = k$$

where *ordered(k)* means *k* is an ordered list. Conditions like *ordered(k)* can often be generated automatically. Moore pioneered this technique in [Moore 1974], and it has been further developed in [Franova and Kodratoff 1992, Monroy, Bundy and Ireland 1994, Protzen 1994]. This technique has the advantage that we can continue with the use of generalisation, instead of having to find an alternative approach.

However, it is not always possible to modify the non-theorem into a theorem which still subsumes the original conjecture. For instance, the conjecture:

$$\forall l: \text{list}(\tau). l <> (l <> l) = (l <> l) <> l \quad (6.12)$$

can be generalised to the non-theorem:

$$\forall l: \text{list}(\tau). k: \text{list}(\tau). l <> k = k <> l$$

This can be modified to the theorem:

$$\forall l: \text{list}(\tau). k: \text{list}(\tau). l = k \rightarrow l <> k = k <> l$$

say, but this no longer subsumes the original conjecture, (6.12).

7. Rippling

Rippling is a difference reduction technique developed for induction proofs. It provides a partial solution to many of the special search control problems described in Section 6, page 865 above. Aubin was the first to notice a common pattern in the rewriting of step cases, [Aubin 1976]. In [Bundy 1988] it was proposed to use this pattern to drive the rewriting process and implementations of this proposal were first reported in [Bundy, van Harmelen, Smaill and Ireland 1990, Bundy, Stevens, van Harmelen, Ireland and Smaill 1993, Hutter 1990].

7.1. Rippling Out

Aubin observed that during the step case the differences between the induction conclusion and the induction hypothesis *ripple-out* of the induction conclusion, leaving a complete copy of the induction hypothesis embedded in the induction conclusion. The effect is emphasised by annotating these differences, *e.g.* by placing them in grey boxes. Consider again the step case of the associativity of $\langle \rangle$ reproduced from Section 4.1.3, page 857, but this time with annotation.

$$\begin{aligned}
 t \langle \rangle (Y \langle \rangle Z) &= (t \langle \rangle Y) \langle \rangle Z \vdash h :: t^{\uparrow} \langle \rangle (y \langle \rangle z) = (h :: t^{\uparrow} \langle \rangle y) \langle \rangle z \\
 &\vdash h :: t \langle \rangle (y \langle \rangle z)^{\uparrow} = h :: t \langle \rangle y)^{\uparrow} \langle \rangle z \\
 &\vdash h :: t \langle \rangle (y \langle \rangle z)^{\uparrow} = h :: (t \langle \rangle y) \langle \rangle z^{\uparrow} \\
 &\vdash h = h \wedge t \langle \rangle (y \langle \rangle z) = (t \langle \rangle y) \langle \rangle z^{\uparrow}
 \end{aligned}$$

The grey boxes indicate the parts of the induction conclusion which differ from the induction hypothesis. They are called *wave-fronts*¹⁸. Each wave-front has one or more *wave-holes* indicating sub-terms of the wave-fronts which correspond to parts of the induction hypothesis. The parts of the induction conclusion outside the wave-fronts or inside the wave-holes, are called the *skeleton*. The skeleton always matches the induction hypothesis. The arrows indicate the direction of movement of the wave-fronts — in this case outwards through the induction conclusion until they completely surround the skeleton. Note how the grey boxes get bigger at each step with more of the skeleton embedded within them, until they contain a complete instance of the induction hypothesis. At this point, strong fertilization can take place.

Rippling restricts the rewriting process so that the skeleton is preserved and wave-fronts are only moved in desirable directions. This is achieved by annotating both the rewrite rules and the induction conclusion and requiring the annotations

¹⁸Note that, to avoid clutter, brackets will be omitted in formulae where the grouping implied by wave-fronts make this redundant, *e.g.* we will write $h :: t^{\uparrow} \langle \rangle (y \langle \rangle z)$ instead of $(h :: t)^{\uparrow} \langle \rangle (y \langle \rangle z)$.

to match. Annotated rewrite rules are called *wave-rules*. The wave-rules required in the example above are:

$$\begin{aligned} H :: T^{\uparrow} <> L &\Rightarrow H :: T <> L^{\uparrow} \\ X_1 :: X_2^{\uparrow} = Y_1 :: Y_2^{\uparrow} &\Rightarrow X_1 = Y_1 \wedge X_2 = Y_2^{\uparrow} \end{aligned}$$

which are annotated versions of rule (4.3) and the replacement rule for $::$. The wave-rules are annotated so that the wave-fronts are further out in the skeleton on the RHS than on the LHS. Any wave-fronts in the redex in the induction conclusion must match corresponding wave-fronts in the LHS of the wave-rule which is applied to it. This last condition reduces the search during rewriting by preventing rewrites in which the annotation does not match.

7.2. Simplification of Wave-Fronts

It is sometimes necessary to apply regular rewrite rules as well as wave-rules during rippling in order to simplify expressions. However, this simplification can be restricted to wave-fronts. The skeleton must be preserved, so must not be rewritten. An example occurs in the *rev-rev* example from Section 6.2.1, page 869. In wave annotation the step case of this proof is:

$$\begin{aligned} rev(rev(t)) &= t \vdash rev(rev(h :: t^{\uparrow})) = h :: t^{\uparrow} \\ \vdash rev(rev(t) <> h :: nil^{\uparrow}) &= h :: t^{\uparrow} \end{aligned} \quad (7.1)$$

$$\vdash rev(h :: nil <> rev(rev(t)))^{\uparrow} = h :: t^{\uparrow} \quad (7.2)$$

$$\vdash rev(nil <> (h :: nil <> rev(rev(t))))^{\uparrow} = h :: t^{\uparrow}$$

$$\vdash nil <> (h :: nil <> rev(rev(t)))^{\uparrow} = h :: t^{\uparrow}$$

$$\vdash (h :: nil <> rev(rev(t)))^{\uparrow} = h :: t^{\uparrow}$$

$$\vdash h :: (nil <> rev(rev(t)))^{\uparrow} = h :: t^{\uparrow} \quad (7.3)$$

$$\vdash h :: rev(rev(t))^{\uparrow} = h :: t^{\uparrow}$$

$$\vdash h = h \wedge rev(rev(t)) = t^{\uparrow}$$

From step (7.2) to step (7.3) no rippling-out takes place, but a wave-front is simplified using rewrite rules from the recursive definitions of *rev* and $<>$, (6.6) and (4.3). Note that the skeleton is not rewritten, since this would jeopardise the potential for fertilization.

This example also illustrates that rippling can be used to guide the application of lemmas as well as recursive definitions. At step (7.1) the lemma (6.7) is applied. This can be annotated as a wave-rule as:

$$\text{rev}(X \langle \rangle Y^\uparrow) \Rightarrow \text{rev}(Y) \langle \rangle \text{rev}(X)^\uparrow \quad (7.4)$$

7.3. Rippling Sideways and In

Rippling wave-fronts right outside the skeleton is one way to enable fertilization, but it is not the only way. We can also exploit the flexibility provided by additional universal variables in the conjecture (see Section 2.3, page 850). These additional variables become free variables in the induction hypothesis and arbitrary constants in the induction conclusion. We will call the arbitrary constants, *sinks*. We can move wave-fronts to surround the sinks. They will then be absorbed by the free variables during fertilization. We will mark sinks thus: $[c]$; you can think of these marks as representing a kitchen sink with a plug hole at the bottom.

To see how this works consider again the example step case from Section 6.2.2, page 870, but this time annotated with wave fronts and sinks.

$$\text{rotate}(\text{length}(t), t \langle \rangle K) = K \langle \rangle t$$

$$\begin{aligned} &\vdash \text{rotate}(\text{length}(h :: t^\uparrow), h :: t^\uparrow \langle \rangle [k]) = [k] \langle \rangle h :: t^\uparrow \\ &\vdash \text{rotate}(s(\text{length}(t))^\uparrow, h :: t \langle \rangle [k])^\uparrow = [k] \langle \rangle h :: t^\uparrow \\ &\vdash \text{rotate}(\text{length}(t), t \langle \rangle [k] \langle \rangle (h :: \text{nil})^\downarrow) = [k] \langle \rangle (h :: t^\uparrow) \end{aligned} \quad (7.5)$$

$$\vdash \text{rotate}(\text{length}(t), t \langle \rangle [k] \langle \rangle (h :: \text{nil})^\downarrow) = [k] \langle \rangle (h :: \text{nil})^\downarrow \langle \rangle t \quad (7.6)$$

$$\vdash \text{rotate}(\text{length}(t), t \langle \rangle [k \langle \rangle (h :: \text{nil})]) = [k \langle \rangle (h :: \text{nil})] \langle \rangle t \quad (7.7)$$

At step (7.5) instead of moving the LHS wave-front further outwards we move it sideways and then inwards towards the sink. We call these processes *rippling-sideways* and *rippling-in*. The inwards direction of this wave-front is indicated by the downwards arrow. At step (7.6) the RHS wave-front also moves sideways and then inwards. When an inwards wave-front immediately dominates a sink, as at step (7.6), then it can be absorbed into the sink. This has been done twice in the last step (7.7). Strong fertilization is now possible with the free variable, K , being matched to the contents of the sink, $k \langle \rangle (h :: \text{nil})$.

To implement rippling-sideways and rippling-in we need wave-rules with a slightly different kind of annotation. The sideways¹⁹ wave-rules used in the above example are annotated as²⁰:

$$\text{rotate}(s(N)^\uparrow, H :: T^\uparrow) \Rightarrow \text{rotate}(N, T \langle \rangle (H :: \text{nil})^\downarrow) \quad (7.8)$$

¹⁹Also called *transverse* wave-rules, in contrast to *longitudinal* wave-rules, which ripple-out.

²⁰See Section 7.9, page 885 for a discussion of the annotation of wave-rule (7.8).

$$L <> (H :: T)^{\uparrow} \Rightarrow L <> (H :: nil)^{\downarrow} <> T$$

Functions defined by tail recursion are a good source of such sideways rules, e.g. the definition of *rotate*. The inwards wave-rule used in the above example is one of the many²¹ annotations of associativity:

$$(X <> Y) <> Z^{\downarrow} \Rightarrow X <> (Y <> Z)^{\downarrow}$$

One of the preconditions of rippling sideways and inwards is that any inwards wave-front should have a target to ripple towards (see Section 7.7.4, page 883) in one of its wave-holes. This can be a sink, as above, or an outwards directed wave-front, with which it can cancel. Without such a target the final fertilization will not be possible. This precondition puts a further restriction on rippling.

7.4. The Advantages of Rippling

Rippling has the following advantages over conventional rewriting:

- It is more restrictive:** The condition that wave annotations match prevents rewritings that would otherwise be allowed. This reduces the size of the search space of rippling compared to that of unrestricted rewriting (see Section 7.5, page 879). This would be a disadvantage if desirable rewritings were disallowed, but experiment shows that this does not happen.
- It always terminates:** A general termination proof can be given for all sets of wave-rules (see Section 7.8, page 884), whereas a separate termination proof has to be given for each set of rewrite rules in conventional rewriting.
- It allows rewriting in both directions:** An equation can often be oriented as a wave-rule in each direction by annotating it in two different ways (see Section 7.6, page 880). These different annotations prevent looping, so that termination is preserved. This allows rewriting in different directions even within the same proof.
- It supports various heuristics:** The failure of rippling can be used to suggest patches to a partial proof which help in the choice of generalisations, lemmas and induction rules (see Section 8, page 890).

7.5. Selective Rewriting

Suppose we had the goal of proving:

$$((c + d) + a) + b = (c + d) + 42 \quad (7.9)$$

from the hypothesis $a + b = 42$ with the aid of the associativity of $+$ rewrite rule:

$$(X + Y) + Z \Rightarrow X + (Y + Z) \quad (7.10)$$

²¹See Section 7.6, page 880 for more ways to annotate associativity.

Using conventional rewriting goal (7.9) can be rewritten with rule (7.10) in three ways:

$$\begin{aligned} ((c + d) + a) + b &= c + (d + 42) \\ (c + (d + a)) + b &= (c + d) + 42 \\ (c + d) + (a + b) &= (c + d) + 42 \end{aligned} \quad (7.11)$$

The first two of these three subgoals make no progress towards the hypothesis and represent unwanted branches of the search space. Only subgoal (7.11) represents progress. The hypothesis can now be applied to it, using weak fertilization, to complete the proof.

Rippling can be applied to this problem by annotating the goal (7.9) and the rewrite rule (7.10) with wave-fronts (see Section 7.9, page 885 for how the annotation process can be automated). (7.9) is annotated so that its skeleton is the hypothesis:

$$((c + d) + a^\uparrow) + b = (c + d) + 42^\uparrow$$

The rule (7.10) can be annotated in several ways, but only one of these permits any rippling with (7.9):

$$(X + Y^\uparrow) + Z \Rightarrow X + (Y + Z)^\uparrow$$

The condition that wave annotations match only allows one ripple — the desired one to:

$$(c + d) + (a + b)^\uparrow = (c + d) + 42^\uparrow$$

Thus, rippling limits the rewriting search space by eliminating unproductive rewritings.

7.6. Bi-Directional Rewriting

Equations can be annotated as wave-rules in more than one way. In particular, an equation can often be annotated in either orientation. For instance, the associativity law of $\langle \rangle$ can be annotated in the following six ways:

$$\begin{aligned} X \langle \rangle (Y \langle \rangle Z)^\uparrow &\Rightarrow (X \langle \rangle Y) \langle \rangle Z^\uparrow \\ X \langle \rangle (Y \langle \rangle Z)^\uparrow &\Rightarrow (X \langle \rangle Y)^\downarrow \langle \rangle Z \\ X \langle \rangle (Y \langle \rangle Z)^\downarrow &\Rightarrow (X \langle \rangle Y)^\downarrow \langle \rangle Z \end{aligned}$$

$$\begin{aligned}
(X <> Y)^\uparrow <> Z &\Rightarrow X <> (Y <> Z)^\uparrow \\
(X <> Y)^\uparrow <> Z &\Rightarrow X <> (Y <> Z)^\downarrow \\
(X <> Y) <> Z^\downarrow &\Rightarrow X <> (Y <> Z)^\downarrow
\end{aligned}$$

The first three are oriented in one direction and the second three are oriented in the other. Moreover, all six wave-rules are measure decreasing, left to right. This means that we could use any combination of them in the same ripple sequence without risk of non-termination. This is a surprising departure from the normal situation in rewriting. Normally using an equation as a rewrite rule in both orientations could cause looping. What prevents rippling from looping is that the wave annotations will prevent the same equation being applied to reverse a previous rewrite, *i.e.* that if you take the meta-functions into account then the equations are not reversible.

This ability to rewrite in either direction is frequently useful. We found a need for it in step (6.9) in Section 6.3.2, page 873. The step case of the generalised *rev-rev* conjecture required a rewrite rule to be applied backwards. If we annotate this step case we can see how rippling can enable this. The annotated step case is:

$$\text{rev}(t' <> (h :: \text{nil})) = h :: \text{rev}(t')$$

$$\vdash \text{rev}(h' :: t')^\uparrow <> (h :: \text{nil}) = h :: \text{rev}(h' :: t')^\uparrow$$

$$\vdash \text{rev}(h' :: t' <> (h :: \text{nil}))^\uparrow = h :: \text{rev}(t') <> (h' :: \text{nil})^\uparrow \quad (7.12)$$

$$\vdash \text{rev}(t' <> (h :: \text{nil})) <> (h' :: \text{nil})^\uparrow = h :: \text{rev}(t') <> (h' :: \text{nil})^\uparrow \quad (7.13)$$

$$\vdash \text{rev}(t' <> (h :: \text{nil})) = h :: \text{rev}(t') \wedge h' :: \text{nil} = h' :: \text{nil}^\uparrow$$

Note that step (7.13) is achieved on the RHS with the wave-rule:

$$H :: T <> L^\uparrow \Rightarrow H :: T <> L^\uparrow$$

which is an annotation of rewrite rule (4.3), but in an inverted orientation. Step (7.12) on the LHS, on the other hand, is achieved by a different annotation of rewrite rule (4.3) in its given orientation, namely:

$$H :: T^\uparrow <> L \Rightarrow H :: T <> L^\uparrow$$

This bi-directional use of the same equation within the same derivation is handled smoothly by rippling without looping.

Examples where the same equation needs to be used in different orientations *within the same proof* are relatively rare (but do happen – see the example above). However, it is very common for the same equation to be used in different orientations within a family of proofs. For instance, associativity and distributivity laws are used

in both orientations quite frequently. Individual problem equations can be built-into the unification algorithm, *e.g.* associativity, but there will always be equations which have not yet been so built-in or which cannot easily be built-in. Rippling gives a useful flexibility in such cases.

7.7. The Definition of Wave Annotation

Wave-rules can be formally defined as annotated rewrite rules which are skeleton preserving and measure decreasing. Full definitions of the concepts of well annotated term, skeleton and measure can be found in [Basin and Walsh 1996], together with a proof of the termination of rippling and an algorithm, called *difference unification*, for annotating formulae. We give an overview of this account here.

7.7.1. Meta-Level Functions

Wave annotations can be thought of as meta-level functions which are inserted into the object-level terms. These meta-functions are:

wf: which defines a wave-front. This meta-function has a second argument of *in* or *out* to indicate the direction of the wave-front.

wh: which defines a wave-hole within a wave-front.

snk: which defines a sink.

So $\text{rotate}(h :: t^\uparrow, [l])$ is represented by $\text{rotate}(wf(h :: wh(t), out), snk(l))$.

7.7.2. Normal Forms and Well-Formedness

It is convenient for both technical and implementational reasons to put annotated terms into a normal form in which wave-fronts are all one-functor²² thick, *i.e.* to split wider wave-fronts into a nested sequence of wave-fronts and wave-holes,

e.g. $s(s(n))^\uparrow$ is put into the normal form $s(s(n)^\uparrow)^\uparrow$. Another part of the normal form is to absorb inward directed wave-fronts into sinks that they immediately dominate, *e.g.* $f(a, [b])^\downarrow$ is rewritten to $[f(a, b)]$.

Let f be a functor immediately dominated by wf . At least one of the arguments of f must then be dominated by a wh , but several can be. For instance, in $f(a, b, c)^\uparrow$ f is dominated by wf and two of its three arguments are dominated by wh . f and b are said to be *in the wave-front* and a and c are said to be *in wave-holes*. It is a condition of well-formedness that any wave-fronts nested inside f must be nested in one of its wave-holes, *i.e.* the following is ill-formed

$f(g(a)^\uparrow, b)^\uparrow$. Sometimes matching inserts a wave-front in one of the non-wave-hole arguments of f . The matcher must delete these meta-functions to make the term well annotated, *i.e.* rewrite the above ill-formed term to²³ $f(g(a), b)^\uparrow$. Apart

²²“Functor” is a synonym for “function symbol”, *e.g.* in the term $f(x)$ f is a functor.

²³This wave-front is still one functor thick; only f dominates the wave-hole b .

from this requirement, matching of the LHS of a wave-rule to a redex is done by the standard matching algorithm with the meta-functions being treated as normal functions. Note that this means that any wave annotation in the LHS must match corresponding wave annotation in the redex and that any wave annotation in the redex must match either a variable in the LHS or corresponding wave annotation there.

7.7.3. Skeletons and Skeleton Preservation

The skeleton is a set of terms formed by deleting all the wave-fronts and their contents, but retaining the contents of the wave-holes. A skeleton is a set because multiple wave-holes in a function give rise to multiple terms when wave-fronts are deleted.

For instance, the skeleton of $rev(X \langle \rangle Y)^\uparrow$ is $\{rev(X), rev(Y)\}$. A *weakening* of an annotated term is one in which all but one wave-hole is deleted from each function. For instance, $rev(X \langle \rangle Y)^\uparrow$ is a weakening of $rev(X \langle \rangle Y)^\uparrow$. The skeletons of weakenings are always singletons, e.g. $\{rev(X)\}$ in the above example.

A defining property of wave rules is that they are skeleton preserving. Skeleton preservation means that the skeleton of the LHS of the wave-rule is a superset of the skeleton of the RHS. Usually, they are equal, but in some cases this is not possible. Consider, for instance, the replacement wave rule for $\langle \rangle$.

$$X_1 \langle \rangle X_2^\uparrow = Y_1 \langle \rangle Y_2^\uparrow \Rightarrow X_1 = Y_1 \wedge X_2 = Y_2^\uparrow$$

The skeleton of the LHS is $\{X_1 = Y_1, X_1 = Y_2, X_2 = Y_1, X_2 = Y_2\}$ but that of the RHS is only $\{X_1 = Y_1, X_2 = Y_2\}$. There is a way of excluding the unwanted elements of the LHS skeleton, in this case, by associating colours with wave-holes, [Yoshida, Bundy, Green, Walsh and Basin 1994]. In this example the wave-rule is viewed as a doubleton whose members have different colours: a red member, $X_1 = Y_1$, and a blue member, $X_2 = Y_2$. The wave-holes in the wave-rule are coloured appropriately, e.g. consider the replacement rule²⁴ for $\langle \rangle$:

$$X_1^{red} \langle \rangle X_2^{blue}^\uparrow = Y_1^{red} \langle \rangle Y_2^{blue}^\uparrow \Rightarrow X_1 = Y_1^{red} \wedge X_2 = Y_2^{blue}^\uparrow$$

and these colours are taken into account in the definition of skeleton to ensure that colours are not mixed. This makes the skeleton of both sides of the wave-rule be $\{X_1 = Y_1, X_2 = Y_2\}$. Note that the $=$ on the LHS is shared between the red and blue skeleton members and must be labelled with the set $\{red, blue\}$. The advantage of this colour labelling is that skeleton preservation in coloured wave-rules now means *equality* of skeletons.

7.7.4. The Preconditions of Rippling

The preconditions of a ripple are as follows:

1. The induction conclusion contains a wave-front.

²⁴The implication is from right to left, see Section 4.1.2, page 857.

2. A wave-rule exists whose LHS matches a redex in the induction conclusion containing this wave-front.
3. If the wave-rule is conditional then the condition can be proven.
4. Any inwards wave-fronts inserted into the induction conclusion contain either a sink or an outwards wave-front in one of their wave-holes.

In Section 8, page 890 we will consider various ways in which these preconditions might fail and what patch might be applied to the proof in each case.

7.8. Termination of Rippling

To prove termination of rippling we need a measure onto a well-founded set and we need to show that each ripple strictly decreases this measure. The intuition behind this measure is that it decreases when outward directed wave-fronts move towards the root of a term and when inwards directed wave-fronts move towards the leaves. [Basin and Walsh 1996] defines a simple measure with this property²⁵. They prove that if the measure of the RHS of each wave-rule is strictly smaller than that of the LHS then the measure of the result of applying this wave-rule will also be strictly smaller than the goal to which it is applied. This means we can restrict our attention to wave-rules when proving termination. We outline the Basin/Walsh measure in three stages.

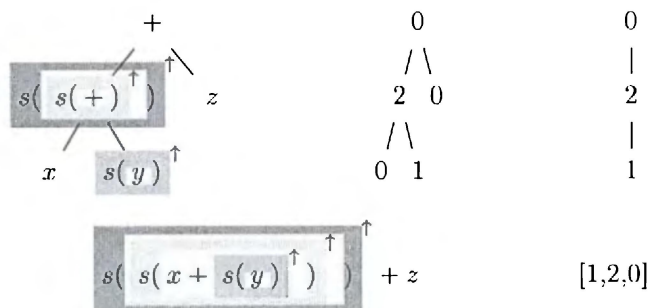
First, consider the case where an annotated term is a weakening, *i.e.* has a singleton skeleton, and has only outwards directed wave-fronts. Consider this skeleton as a parse tree with each node labelled by the wave-fronts immediately dominating that functor in the skeleton. An example is given in figure 2. Now abstract this parse tree by replacing all the labels with the weight of the wave-fronts at that point in the tree. There are various ways to calculate the weight, but the one we will use is just the number of wave-fronts. Finally, we make a list where each element corresponds to the depth of the tree and contains the total weight of wave-fronts at that depth. Such lists can be well ordered by the lexicographic order in which the element at greatest depth has highest precedence. In figure 3 is an example showing how the measure decreases during rippling.

Secondly, consider the case where a term has a non-singleton skeleton, but still contains only outwards wave-fronts. The measure of this term is the multi-set of the measures of each of its weakenings, ordered by the multi-set ordering. For instance, the measure of $rev(X \langle \rangle Y)$ is $\{[1, 0], [1, 0]\}$.

Thirdly, consider the case where a term contains a mixture of outwards and inwards wave-fronts. We define an inwards measure exactly like the outwards one, but with the lists lexicographically ordered in the reverse direction, *i.e.* with the element at least depth having highest precedence. For instance, the inwards measure of $rev(X \langle \rangle Y)$ is $\{[0, 1], [0, 1]\}$.

The overall measure is the lexicographically ordered pair of the outwards and inwards measures, with the outwards measure having precedence over the inwards

²⁵A variety of other measures with this property are possible.



In the bottom left hand corner is the term whose measure is to be calculated. In the top left hand diagram the wave-front is used to label the parse tree of the skeleton. In the middle diagram the node labels are abstracted to show just the weight of the wave-fronts at that point. In the top right hand diagram the parse tree is replaced by a list with each element showing the total weight of wave-fronts at that depth. This list is reproduced in the standard horizontal format at bottom right.

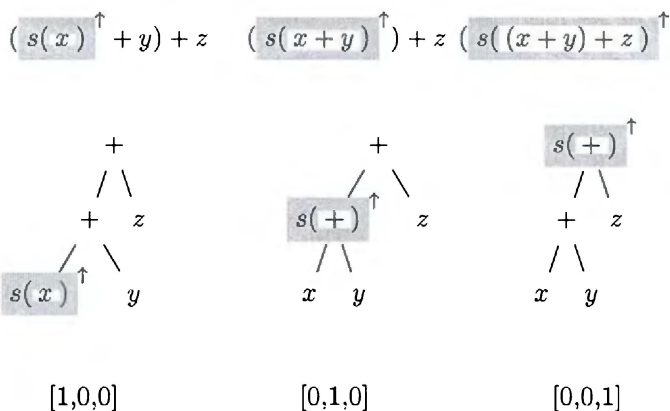
Figure 2: The Outwards Measure of Annotated Terms

one. For instance, the overall measure of $\text{length}(h :: t^\uparrow + s(s(\text{dble}([x])))^\downarrow)$ is $\{\{[1, 0, 0]\}, \{[0, 2, 0]\}\}$. One consequence of the outwards measure having precedence is that just reversing the direction of an outwards wave-front to direct it inwards will result in a measure decrease, but not vice versa. Under this overall measure rippling can be shown to always terminate.

7.9. Automatic Annotation

Terms can be automatically annotated by a process called *difference unification*, [Basin and Walsh 1993]. This is like regular unification but there is an additional option to hide structure in wave-fronts. The *conflict* rule of unification fails the unification attempt if the outermost functions of the unificands are not identical. In difference unification the conflict rule is replaced with two hide rules: one to hide the mismatching function on the left and one to hide the one on the right. The choice of hiding rules makes difference unification non-deterministic; in general, it returns several difference unifiers. If hiding is only allowed on one side we have *difference matching*. If no instantiation of variables is allowed then we have *ground difference unification*. Wave-rules and induction rules can be annotated by ground difference unification.

Directions of wave-fronts can then be inserted by a generate and test process;



In the top row are three annotated terms in successive stages of a ripple. In the middle row these three terms are each represented by the parse trees of their skeletons annotated by the wave-fronts at each node. In the bottom row are the measures of these three terms. Note that under the lexicographic order each measure is strictly less than the one before.

Figure 3: The Strict Decrease of the Outward Measure

each possible combination of directions is tested for measure decrease. Because the outwards measure is lexicographically ordered before the inwards one it is always possible to obtain a measure decrease in a wave-rule by directing LHS wave-fronts outwards and RHS wave-fronts inwards. In order to prevent over production of wave-rules it is usual to restrict this device to those situations where it is strictly necessary to enable a legal annotation. For instance, if difference unification has found and inserted the following wave-fronts in the associative law of $\langle \rangle$:

$$(X \langle \rangle Y) \langle \rangle Z \Rightarrow X \langle \rangle (Y \langle \rangle Z)$$

then the only way that directions can be added to these wave-fronts to create a measure decrease is:

$$(X \langle \rangle Y^\uparrow) \langle \rangle Z \Rightarrow X \langle \rangle (Y \langle \rangle Z^\downarrow)$$

However, the following wave-fronts of the step case of the *rotate* function:

$$\text{rotate}(s(N), H :: T) \Rightarrow \text{rotate}(N, T \langle \rangle (H :: \text{nil}))$$

can be directed, for instance, as:

$$\text{rotate}(s(N)^\uparrow, H :: T^\uparrow) \Rightarrow \text{rotate}(N, T \langle \rangle (H :: \text{nil})^\uparrow)$$

It is not necessary to direct the RHS wave-front inwards to get measure decrease, *i.e.*

$$\text{rotate}(s(N)^\uparrow, H :: T^\uparrow) \Rightarrow \text{rotate}(N, T <> (H :: \text{nil})^\downarrow)$$

But sometimes we *do* want a wave-rule of this form (see, for instance, wave-rule (7.8) in Section 7.3, page 878). To make such annotations available we can either remove the restriction on non-minimal wave-rule annotations and allow (7.8) as a legal annotation or we can employ a meta-rule that any outwards wave-front in the induction conclusion can be turned inwards provided it contains a sink or an outwards wave-front in one of its wave-holes. Of course, we cannot turn inwards wave-fronts outwards since this would usually increase the measure.

There is a similar choice over weakenings. Sometimes weakened forms of wave-rules are required for a proof. Consider, for instance, the use of wave-rule (7.4) in Section 7.2, page 877.

$$\text{rev}(X <> Y^\uparrow) \Rightarrow \text{rev}(Y) <> \text{rev}(X)^\uparrow$$

This wave-rule is a weakening of:

$$\text{rev}(X <> Y^\uparrow) \Rightarrow \text{rev}(Y) <> \text{rev}(X)^\uparrow$$

We can either generate such weakenings explicitly or adapt wave-rule matching to automatically weaken the wave-rule as required. This is a space/time tradeoff.

7.10. Ripple Analysis

Rippling suggests a useful alternative to recursion analysis (see Section 6.1.1, page 866) for providing initial induction rule suggestions to prove a conjecture. In recursion analysis we use the recursive arguments of functions in conjectures to suggest induction rules. In *ripple analysis* we look ahead into the rippling process to see which induction rules will support the initial stage of rippling. This will allow us to use any argument of a function, *provided there is a wave rule in which this argument contains a wave-front*. Since recursive definitions provide wave-rules in which the recursive arguments contain wave-fronts, ripple analysis includes but extends recursion analysis.

To see how this works, consider the conjecture:

$$\forall y:\tau, xs:\text{list}(\tau). y \oplus \text{foldleft}(\oplus, e, xs) = \text{foldleft}(\oplus, y, xs) \quad (7.14)$$

where *foldleft* is a functional defined by:

$$\begin{aligned} \text{foldleft}(F, X, \text{nil}) &\Rightarrow X \\ \text{foldleft}(F, X, H :: T^\uparrow) &\Rightarrow \text{foldleft}(F, F(X, H)^\downarrow, T) \end{aligned} \quad (7.15)$$

and \oplus is an associative function with a right identity e :

$$\begin{aligned} X \oplus (Y \oplus Z^\uparrow) &\Rightarrow (X \oplus Y) \oplus Z^\uparrow \\ X \oplus e &\Rightarrow X \end{aligned} \quad (7.16)$$

Assume, in addition, that the following wave-rule is also available as a lemma:

$$foldleft(F, Z, L <> (X :: nil)^\uparrow) \Rightarrow F(foldleft(F, Z, L), X)^\uparrow \quad (7.17)$$

Since *foldleft* recurses on its third argument, xs is unflawed in (7.14). Thus recursion analysis will suggest a one-step induction on xs using (4.6). However, this induction does not lead to fertilization. The step case will proceed as follows:

$$Y \oplus foldleft(\oplus, e, xs) = foldleft(\oplus, Y, xs)$$

$$\vdash [y] \oplus foldleft(\oplus, e, h :: xs^\uparrow) = foldleft(\oplus, [y], h :: xs^\uparrow) \quad (7.18)$$

$$\vdash [y] \oplus \underbrace{foldleft(\oplus, e, h :: xs^\uparrow)}_{\text{blocked}} = foldleft(\oplus, [y \oplus h], xs) \quad (7.19)$$

at which point the left-hand side wave-front is blocked because there is no sink in the second argument of *foldleft*. Weak fertilization can take place to yield:

$$y \oplus foldleft(\oplus, e, h :: xs) = (y \oplus h) \oplus foldleft(\oplus, e, xs)$$

but again one-step induction on xs is suggested and this time no rippling is possible, resulting in a failed proof.

We now consider how rippling analysis will deal with this example. We look ahead into the rippling process and for each combination of occurrences of universally quantified variables in the conjecture we ask *if they were replaced by suitable wave-fronts, whether a wave-rule would then apply*. Consider, for instance, the second occurrence of xs in (7.14). If this occurrence of xs were replaced by $h :: xs^\uparrow$ then wave-rule (7.15) from the recursive definition of *foldleft* would apply to (7.14), as at step (7.18) above. So the second occurrence of xs suggests an induction on xs using the one-step list induction (4.6). To implement this ripple analysis process efficiently we can invert the reasoning described above, *i.e.* we can use the available wave-rules to suggest which combinations of variables to replace with which wave-fronts, so that those wave-rules will apply.

So far, this reasoning merely recapitulates recursion analysis in different terminology. The first difference comes when we consider the first occurrence of xs . Under recursion analysis this also suggested induction rule (4.6), since it also occurred in the recursion argument of *foldleft*. However, under ripple analysis, if this occurrence of xs were replaced by $h :: xs^\uparrow$ then wave-rule (7.15) would *not* apply, but would be blocked due to the absence of an appropriate sink, as in step (7.19) above.

The second difference with recursion analysis, is that ripple analysis can use lemma (7.17) with both occurrences of xs to suggest a different induction rule. If either occurrence of xs is replaced by $xs \lt (x :: nil)^\uparrow$ then wave-rule (7.17) will apply. This wave-front suggests the induction rule.

$$\frac{P(nil), \quad \forall x:\tau, l: list(\tau). P(l) \rightarrow P(l \lt (x :: nil)^\uparrow)}{\forall l: list(\tau). P(l)} \quad (7.20)$$

Since both occurrences of xs suggest induction rule (7.20) and only one occurrence suggests induction rule (4.6) then rule (7.20) is preferred. Under this rule the step case is successful:

$$\begin{aligned} [y] \odot foldleft(\oplus, e, xs \lt (x :: nil)^\uparrow) &= foldleft(\oplus, [y], xs \lt (x :: nil)^\uparrow) \\ [y] \odot (foldleft(\oplus, e, xs) \oplus x)^\uparrow &= foldleft(\oplus, [y], xs) \oplus x^\uparrow \\ ([y] \oplus foldleft(\oplus, e, xs)) \oplus x^\uparrow &= foldleft(\oplus, [y], xs) \oplus x^\uparrow \\ [y] \oplus foldleft(\oplus, e, xs) &= foldleft(\oplus, [y], xs) \end{aligned}$$

using two applications of lemma (7.17), one of associativity law (7.16) and the replacement rule for \oplus . Strong fertilization is now possible.

Recursion analysis suggests induction rules dual to the recursive definitions of functions in the conjecture. Ripple analysis uses the available wave-rules to suggest induction variables and rules. The wave-fronts in these wave-rules suggest the form of induction. In example (7.14), $xs \lt (x :: nil)^\uparrow$ was used to replace both occurrences of xs . This is the wave-front which occurs in induction rule (7.20). For ripple analysis to recover the appropriate induction rule, each induction rule must be indexed by the wave-fronts in its induction term, e.g. (7.20) must be indexed by $xs \lt (x :: nil)^\uparrow$. Unfortunately, no-one has yet developed a mechanism which given an induction term creates a corresponding induction rules. So, in our example, if rule (7.20) is not already pre-stored then it cannot be used. We return to this issue in Section 9.4, page 897.

So, just as in recursion analysis, induction rules must be constructed from the termination proofs of recursive functions in conjectures. In addition the induction hypotheses and induction conclusions of these induction rules must be difference matched and annotated with wave-fronts. The induction rules must then be indexed by the wave-fronts they contain, so that ripple analysis can access induction rules containing appropriate wave-fronts. Induction rules created by other means, e.g. provided by a user, must be indexed in a similar way.

Ripple analysis, like recursion analysis, only supplies the initial induction rule suggestions. Where these suggestions are incompatible it may be necessary to reject inferior suggestions and combine the remainder using the techniques described in Section 6.1.4, page 868 for recursion analysis.

8. The Productive Use of Failure

The discussion of search control problems in Section 6, page 865 identified lots of places where guidance was needed during inductive proofs. For instance, when is it necessary to introduce a lemma or generalisation and which lemma or generalisation should be used? One successful approach to inductive search control is: to detect when a proof attempt is breaking down; analyse the cause of the failure; and use this analysis to direct the search process. This approach is usually called *the productive use of failure*. See [Ireland 1992, Ireland and Bundy 1996b], for instance, for more discussion of this approach.

In order to detect proof failure you have to have a strong expectation of how it *should* have gone. Such a strong expectation is provided, for instance, by rippling. So we will illustrate the detection, analysis and correction of failure with two examples based on the breakdown of rippling. In both cases the initial proof attempt has reached the step case of an inductive proof and rippling has been initiated. It has then failed because the preconditions of a particular ripple (see Section 7.7.4, page 883) were not met. Differences in the precondition failure, however, suggest a different proof patch in each case.

8.1. Example: Speculating a Lemma

Consider again the generalised rotate length conjecture, (6.8) from Section 6.2.2, page 870. We saw how an attempt to prove this conjecture without lemmas would get stuck in the step case after the following ripple:

$$\text{rotate}(\text{length}(t), t \<> K) = K \<> t$$

$$\vdash \text{rotate}(\text{length}(h :: t^\uparrow), h :: t^\uparrow \<> [k]) = [k] \<> h :: t^\uparrow$$

$$\vdash \text{rotate}(s(\text{length}(t))^\uparrow, h :: t \<> [k])^\uparrow = [k] \<> h :: t^\uparrow$$

$$\vdash \text{rotate}(\text{length}(t), t \<> [k] \<> (h :: \text{nil})^\downarrow) = [k] \<> (h :: t^\uparrow)$$

At this point no further wave-rules apply.

In terms of the preconditions of rippling we have the following situation:

1. Both sides of the induction conclusion contain wave-fronts.
2. No wave-rule exists whose LHS matches any redex in the induction conclusion containing these wave-fronts. There isn't even a near match. *So this precondition fails.*
3. With no wave-rule there is no condition to check.
4. Similarly, there are no inwards wave-fronts to check.

In such cases the best bet seems to be to introduce a lemma which can be annotated as the missing wave-rules. We can say a lot about the structure of this missing wave rule. For instance, on the LHS of our example above the redex we

want to rewrite is: $(t \langle \rangle [k]) \langle \rangle (h :: nil)^\downarrow$. So the LHS of the missing wave-rule must have the form: $(X \langle \rangle Y) \langle \rangle Z^\downarrow$. Note that we must preserve all the structure of the skeleton, but can generalise the contents of the wave-front to a new variable. We can also say a lot about the RHS of the missing wave-rule. It should have the same skeleton as the LHS, but the wave-front can take any form. We can represent this uncertainty about the wave-front by using a second-order meta-function, F , to represent it. Since the point of this wave-rule is to ripple the wave-front towards the sink, $[k]$, we can say that the wave-hole of F should be Y , the free variable that will match $[k]$. So we can speculate the missing wave-rule to be:

$$\exists F. \forall X, Y, Z. (X \langle \rangle Y) \langle \rangle Z^\downarrow \Rightarrow X \langle \rangle F(Y, Z)^\downarrow \quad (8.1)$$

Quantifiers have been inserted to clarify the status of the variables in the proof, but types have been omitted to facilitate readability.

(8.1) can now be fed to the inductive theorem prover as a new conjecture. The proof of conjectures containing second-order meta-functions requires special treatment. In particular, instead of using rewriting we need to use narrowing, *i.e.* rewriting in which free variables in the redex can be instantiated by unification with the rewrite rule. It will also be necessary to use *second-order* unification during narrowing. Note that universal variables like X , Y and Z should *not* be instantiated, but existential variables like F can be. During the proof of (8.1) the second-order variable F is instantiated to $\langle \rangle$ by second-order unification. A small amount of search is introduced by the presence of meta-variables such as F , but this is insufficient to prevent automation. The instantiation of F instantiates the missing rule to the associativity of $\langle \rangle$ annotated as:

$$(X \langle \rangle Y) \langle \rangle Z^\downarrow \Rightarrow X \langle \rangle (Y \langle \rangle Z)^\downarrow \quad (8.2)$$

as expected.

Similar reasoning will speculate the missing RHS wave-rule as:

$$L \langle \rangle (H :: T)^\uparrow \Rightarrow G(L, H)^\downarrow \langle \rangle T$$

during the proof of which G is instantiated to reveal the wave-rule as:

$$L \langle \rangle (H :: T)^\uparrow \Rightarrow (L \langle \rangle (H :: nil)^\downarrow) \langle \rangle T \quad (8.3)$$

again, as expected.

This lemma speculation mechanism can also be used to suggest the missing wave-rule (6.7) from Section 6.2.1, page 869. Analysis of the stuck ripple suggests that the form of the missing wave-rule is:

$$rev(X \langle \rangle Y)^\uparrow \Rightarrow F(X, Y, rev(X))^\uparrow$$

The meta-variable F will be instantiated during subsequent proof to: $F(X, Y, Z) = \text{rev}(Y) \text{ <> } Z$, so that the missing wave-rule is revealed as:

$$\text{rev}(X \text{ <> } Y)^\uparrow \Rightarrow \text{rev}(Y) \text{ <> } \text{rev}(X)^\uparrow$$

as required.

Second and higher-order unification algorithms are non-deterministic. The branching rate can be very high and can cause severe search problems. In this application we can exploit the wave annotations to reduce the branching significantly, i.e. we insist that wave-fronts unify with wave-fronts and skeletons with skeletons. These additional constraints make the lemma speculation technique tractable in many practical cases. [Hutter and Kohlhasse 1997] describes a higher-order unification algorithm for annotated terms which embeds these additional constraints.

In addition, the termination of rippling is lost when meta-variables are present in the conclusion. The search control must avoid infinite branches, e.g. by some element of parallelism in the search using breadth-first or iterative deepening, and by using eager fertilization to terminate branches whenever this is possible.

8.2. Example: Introducing a Sink

Now consider the rotate length conjecture, (6.10), from Section 6.3.3, page 874. An attempt to prove this conjecture using rippling will get stuck in the step case after the following ripple:

$$\begin{aligned} \text{rotate}(\text{length}(t), t) = t \vdash \text{rotate}(\text{length}(h :: t)^\uparrow, h :: t)^\uparrow &= h :: t^\uparrow \\ \vdash \text{rotate}(s(\text{length}(t))^\uparrow, h :: t)^\uparrow &= h :: t^\uparrow \end{aligned}$$

At this point no further wave-rules apply.

In terms of the preconditions of rippling we have the following situation:

1. Both sides of the induction conclusion contain wave-fronts.
2. A wave-rule exists whose LHS matches a redex containing both the LHS wave-fronts, namely:

$$\text{rotate}(s(N)^\uparrow, H :: T)^\uparrow \Rightarrow \text{rotate}(N, T \text{ <> } (H :: \text{nil})^\downarrow)$$

3. The wave-rule is unconditional so there is no condition to prove.
4. The inwards wave-front inserted into the induction conclusion would be $t \text{ <> } (h :: \text{nil})^\downarrow$. This contains neither a sink nor an outwards wave-front in its wave-hole. *So this precondition fails.*

Since we have a matching wave-rule already, there seems little point in looking for another one, until we have tried harder to make the existing one applicable. What is preventing it from applying is the absence of a sink or an outwards wave-front in the appropriate place. So in such cases we should try to insert one of these, starting

with a sink. Sinks are created by the presence of additional universal variables in the conjecture. So this analysis suggests generalising the theorem to introduce an additional universal variable.

The original conjecture is:

$$\forall l : \text{list}(\tau). \text{rotate}(\text{length}(l), l) = l$$

We need a sink in the second argument of *rotate*. Since we don't know how it is attached to the existing argument we can link it with a meta-variable, *i.e.*

$$\forall l, k : \text{list}(\tau). \text{rotate}(\text{length}(l), F(l, k)) = l$$

To balance up this conjecture we had better add the new variable to the RHS too.

$$\forall l, k : \text{list}(\tau). \text{rotate}(\text{length}(l), F(k, l)) = G(k, l)$$

We can now prove this generalised conjecture using narrowing with second-order unification to instantiate *F* and *G*. Assuming that the lemmas (8.2) and (8.3) are available the step case of the proof proceeds as follows:

$$\text{rotate}(\text{length}(t), F(K, t)) = G(K, t)$$

$$\vdash \text{rotate}(\text{length}(h :: t^\uparrow), F([k], h :: t^\uparrow)) = G([k], h :: t^\uparrow)$$

$$\vdash \text{rotate}(s(\text{length}(t))^\uparrow, h :: t <> F_2([k], h :: t^\uparrow)^\uparrow) = G([k], h :: t^\uparrow)$$

$$\vdash \text{rotate}(\text{length}(t), t <> F_2([k], h :: t^\uparrow) <> (h :: \text{nil})^\downarrow) = \\ (G_2([k], h :: t^\uparrow)^\downarrow <> (h :: \text{nil})^\downarrow) <> t$$

where $F(k, l) = l <> F_2(k, l)$ and $G(k, l) = G_2(k, l) <> l$. These instantiations are made by second-order unification during the application of wave-rules (4.3) and (8.3), respectively. The step can now be completed by strong fertilization, with F_2 and G_2 both being instantiated to projection functions onto their first arguments in the process. These instantiations of the meta-functions reveal the generalised conjecture to be:

$$\forall l : \text{list}(\tau). \forall k : \text{list}(\tau). \text{rotate}(\text{length}(l), l <> k) = k <> l$$

as expected.

As with lemma speculation (see Section 8.1, page 890) the presence of these meta-functions creates branch points in the proof search, but the extra constraints provided by the wave annotation reduce the search and make it tractable in many practical cases. We must also take care to avoid infinite regress in the rippling search process.

9. Existential Theorems

The discussion so far has mostly been restricted to conjectures containing only universal variables (see Section 1.2, page 848). Dealing with conjectures which include existential variables requires extending the techniques described above.

9.1. Synthesis Problems

Existential variables are required to represent synthesis problems as theorem proving problems. For instance, suppose the task of sorting a list has been specified as producing an ordered permutation of the original list. The problem of synthesising a sorting algorithm can be represented as the conjecture:

$$\forall l : \text{list}(\tau). \exists k : \text{list}(\tau). \text{ordered}(k) \wedge \text{perm}(l, k)$$

where $\text{ordered}(k)$ means k is ordered and $\text{perm}(l, k)$ means k is a permutation of l . If this conjecture is proved in a constructive logic then a program for sorting lists can be recovered from the proof. Inductive proof is required to introduce recursion or loops into these synthesised programs. Various techniques have been devised for extracting the synthesised program from the proof, but the simplest is as the witness of the existential²⁶ variable k , i.e. during the proof k will be instantiated to a term $\text{sort}(l)$ and the proof will ensure that:

$$\forall l : \text{list}(\tau). \text{ordered}(\text{sort}(l)) \wedge \text{perm}(l, \text{sort}(l))$$

The synthesis proof of sort will require induction and this will cause sort to be defined recursively: the form of induction determining the form of recursion. Different proofs of the theorem will synthesise different algorithms for the same function, e.g. bubble-sort, merge-sort, quick-sort, etc (see [Darlington 1978] for a detailed discussion).

Synthesis of recursively defined software, hardware, etc is an important application of inductive theorem proving. So it is important that inductive theorem proving techniques can handle existential variables, in particular, conjectures of the form:

$$\forall \vec{i} : \vec{\tau}. \exists o : \tau'. \text{spec}(\vec{i}, o)$$

where $\text{spec}(\vec{i}, o)$ specifies the relationship between the inputs, \vec{i} , and the output, o , of the object to be synthesised. Note that spec may contain further quantifiers. Unfortunately, automated synthesis is an area where current technology is weak.

9.2. Representing Existential Theorems

There are a variety of techniques for representing existential variables during automated proof.

²⁶Note that dual skolemisation is applied to conjectures, so that k becomes a free variable (see Section 1.2, page 848).

9.2.1. Existential Variables as First-Order Free Variables

The classic technique, which is standard in resolution theorem proving, for instance, is to dual skolemise the conjecture, replacing universal variables with skolem functions and existential variables with free variables. So our *sort* example will become:

$$\text{ordered}(K) \wedge \text{perm}(l, K)$$

This conjecture will then be proved with first-order unification instead of matching, so that K can be instantiated as a side effect of the proof. At the end of the proof K will be instantiated to $\text{sort}(l)$ and a recursive definition of *sort* will be extracted from the inductive proof.

9.2.2. Existential Variables as Second-Order Free Variables

An equivalent²⁷ formulation of the sorting algorithm synthesis problem is as the second-order conjecture:

$$\exists f : \text{list}(\tau) \mapsto \text{list}(\tau). \forall l : \text{list}(\tau). \text{ordered}(f(l)) \wedge \text{perm}(l, f(l))$$

This can be dual skolemised to:

$$\text{ordered}(F(l)) \wedge \text{perm}(l, F(l))$$

and second-order unification used to instantiate F to *sort* (see Section 8.1, page 890 and Section 8.2, page 892 for examples of this technique). Again the recursive definition of *sort* must be extracted from the proof.

9.2.3. Existential Variables as Skolem Functions

Notice that these techniques of instantiating free variables during the proof do not buy us very much. The variable is merely instantiated to the name of the synthesised object, *e.g.* *sort*, and most of the work of extracting the recursive definition of this object remains. So we might as well do the instantiation at the outset, *i.e.* prove the conjecture:

$$\forall \vec{i} : \vec{\tau}. \text{spec}(\vec{i}, \text{prog}(\vec{i}))$$

where *prog* is the object to be synthesised. This technique was originally developed by Biundo, [Biundo 1988]. We then need to extract a recursive definition of *prog* from the inductive proof.

9.3. Extracting Recursive Definitions

We discuss two techniques for extracting programs from proofs.

²⁷Modulo the axiom of choice.

9.3.1. *Proofs as Programs*

The *proofs as programs* technique was designed for the extraction of programs from synthesis proofs. It uses a constructive type theory, like that due to Martin-Löf, [Martin-Löf 1979] and implemented in NUPRL, [Constable, Allen, Bromley et al. 1986]. From our viewpoint the idea is to associate with each rule of inference, a program construction rule. Initially, the program is represented by a free variable. Each time the prover applies a rule of inference the program is instantiated by the associated program construction rule. This instantiation usually introduces further free variables which are instantiated by subsequent proof steps. At the end of the proof the program can be read off as the instantiation of the original free variable.

The proofs as programs technique is based on the Curry-Howard isomorphism, [Howard 1980], which draws on an analogy between logical rules and type construction rules. Specifications are represented as types and programs meeting these specifications as members of those types, *i.e.* $prog : spec$. The logical rules manipulate the types and the program construction rules manipulate their members (roughly speaking). Both parts are based on a sequent calculus presentation of λ calculus. Higher-order functions and λ abstraction both play an essential role. They are needed in some of the tricky manipulations required in the program construction rules, especially the rules that construct recursive programs from induction proof steps. The program associated with the induction hypothesis must be embedded as the recursive call in the program associated with the induction conclusion. This embedding is neatly done in Martin-Löf Type Theory by representing recursion with recursive functionals, *i.e.* higher-order functions which create recursive functions from their defining functions. The extracted program is a λ calculus function which can be interpreted as a program in a functional programming language. Proofs as programs can also be adapted to the synthesis of hardware and other kinds of objects.

9.3.2. *The Speculation of Program Definitions*

An alternative approach to synthesis is to try to recognise definition-like subgoals during the synthesis proof and convert them into program definitions. These definitions can then be used to complete the proof and to define the synthesised program. This has been explored in different forms by Biundo, [Biundo 1988], and Kraan, [Kraan, Basin and Bundy 1996]. We illustrate the general idea by adapting the technique of lemma speculation of Section 8.1, page 890 using the skolem function representation of Section 9.2.3, page 895 on the *sort* example.

We start with the synthesis conjecture:

$$\forall l : list(\tau). ordered(sort(l)) \wedge perm(l, sort(l))$$

We cannot prove this because we lack a definition of *sort*. This lack may manifest itself during the course of the proof attempt by the failure of rippling. Using the techniques of Section 8.1, page 890 we may speculate the wave-rule:

$$sort(H :: T)^\uparrow \Rightarrow F(H, sort(T))^\uparrow$$

Instead of trying to prove this we can adopt it as the step case of the recursive definition of *sort*. The second-order, meta-variable F can be instantiated to a constant and becomes a new program to be synthesised by the remainder of the synthesis proof. If we instantiate F to *insert*, say, then the partial definition of *sort* is:

$$\text{sort}(H :: T) = \text{insert}(H, \text{sort}(T))$$

9.4. Problems with Recursion Analysis

If recursion analysis (see Section 6.1.1, page 866) is used to construct the induction rule in the synthesis proof of a recursive program then we run into the following problem. The form of induction constructed is based on the forms of recursion in the functions in the conjecture. These functions are all drawn from the specification of the program. The induction rule used will determine the recursive structure of the synthesised program, and thus its essential algorithmic structure. This means that the algorithmic structure of the synthesised program is already implicitly present in its specification.

This puts a limit on the practical creative power of synthesis by proof. The technique cannot break out of the circle of forms of recursion known to it, except by combination and merging of existing forms of recursion. Something radically new (as, for instance, quick-sort historically was) cannot be built without user assistance. Moreover, it is necessary to include algorithmic content in specifications, in the sense that they must include functions with essentially the same kind of recursion as needed in the synthesised program.

Ripple analysis (see Section 7.10, page 887) gives a pointer as to how to break out of this circle. In ripple analysis, induction rules are cued on the basis of wave-fronts in known wave-rules, not recursive functions in the specification. These wave-fronts need not (and often do not) index an induction rule dual to any recursion present in the specification. This frees specifications from being algorithmic. Consider, for instance, the problem of synthesising *quicksort* from the specification:

$$\forall l:\text{list}(\tau). \exists k:\text{list}(\tau). \text{ordered}(k) \wedge \text{perm}(l, k)$$

where *ordered* and *perm* are defined as:

$$\text{ordered}(\text{nil}) \leftrightarrow \top$$

$$\text{ordered}(H :: \text{nil}) \leftrightarrow \top$$

$$\text{ordered}(H_1 :: (H_2 :: T)) \leftrightarrow H_1 \leq H_2 \wedge \text{ordered}(H_2 :: T)$$

$$\text{perm}(\text{nil}, L) \leftrightarrow L = \text{nil}$$

$$\text{perm}(H :: T, L) \leftrightarrow \text{perm}(T, \text{delete}(H, L))$$

where *delete*(H, L) deletes one copy of H from L . Note that recursion analysis will suggest a simple two-step structural induction rule, leading to a similar two-step recursion in the synthesised sorting algorithm, which is not what is required.

Ripple analysis, on the other hand, could suggest the correct form of induction for this synthesis problem using the wave-rule:

$$\text{ordered}(\text{lesseq}(H, T) \text{ } <> (H :: \text{greater}(H, T))^\uparrow) \Rightarrow \text{ordered}(T) \quad (9.1)$$

assuming it was available as a lemma²⁸.

Suppose the wave-front is just used to index an induction rule already known to the prover, *i.e.* in practice, one that has arisen from the termination proof of a known recursive function. The synthesis technique will then not be able to break out of the circle of known forms of recursion and simple combinations of them. To construct a radically new form of recursion it is necessary to synthesise new induction rules from wave-fronts. In our example it would be necessary to use the wave-fronts in wave-rule (9.1) to synthesise the special-purpose induction rule:

$$\frac{P(\text{nil}) \quad \forall h:\tau. \forall t:\text{list}(\tau). P(\text{lesseq}(h, t)) \wedge P(\text{greater}(h, t)) \rightarrow P(h :: t)}{\forall l:\text{list}(\tau). P(l)}$$

which is a special case of Noetherian induction, Section 2.1, page 849, where $<$ is the relationship of one list being shorter than another. This problem has been addressed by Protzen, [Protzen 1994], who uses Walther's techniques for proving termination, [Walther 1994b], to construct a $<$ from the wave-front and hence construct an induction rule customised for the conjecture. In general, synthesising induction rules from wave-fronts is a hard problem (since it embeds the halting problem) and only partial solutions are possible.

10. Interactive Theorem Proving

The difficulty of the search control problems that arise in inductive theorem proving means that all current automatic provers fail on some apparently simple conjectures. Even totally automatic provers are often sensitive to the precise definitions of functions in, parameterisations of or lemmas available to the prover. Until the technology is significantly improved it is, therefore, necessary to involve a human user in assisting with proof search.

10.1. Division of Labour

There is a continuum from purely interactive to purely automatic provers, and most provers lie somewhere in the middle of this continuum; routine proof tasks are automated and hard proof tasks require human interaction. Examples of routine tasks which are often automated are: keeping track of the state of the proof; matching and unification of expressions; the simplification of expressions; the application of decision procedures; and the exhaustive application of a set of rewrite rules. Typically, these require the application of a straightforward algorithm, so are easy to

²⁸Which is, admittedly, a strong assumption

automate, but are long-winded manipulations in which humans can easily become lost or make errors. Examples of hard tasks which are sometimes left to human interaction are: the choice of induction rule; the decision to split into cases; the application of a lemma; and the generalisation of the conjecture. Typically, these involve a crucial search decision or construction of a key expression which require some insight into the structure of the proof.

10.2. *Tactic-Based Provers*

A popular framework for semi-automated theorem proving is the use of *tactics*. A tactic is a computer program for guiding the proof search. This program may apply a rule of inference or combine two or more tactic applications using *tacticals*. There are tacticals for successive application, repeated application, conditional application, *etc.* Tactics are constructed for a variety of routine tasks, *e.g.* simplification of expressions, applying decision procedures, applying sets of rewrite rules, applying induction, generalising formulae, *etc.* The user can then direct the proof search either by calling individual rules of inference or by calling a tactic, which will apply several rules of inference. Much of the tedium and error is thus removed from the interactive process. The user may assist the tactic application by providing key parameters, *e.g.* which induction rule to use, which formula to generalise the current conjecture to. The user can view the proof either at the high level of tactic applications or at the low level of individual rules.

Tactics were invented by Milner and his co-workers and first implemented in the LCF system, [Gordon, Milner and Wadsworth 1979]. They developed the ML (Meta-Language) functional programming language to describe tactics in LCF. Each tactic is an ML program which can construct new theorems from old ones. ML uses types to ensure the soundness of the tactics. "Theorem" is an ML type; an expression cannot be of type theorem unless it is the result of a proof. A whole family of tactic-based provers have been built in the LCF tradition, including Coq, HOL, Isabelle, NuPrl and *Oyster*.

10.3. *User Interfaces*

To enable users to guide semi-automated inductive provers it is necessary to provide a user interface. Such interfaces need to be designed with the problems of inductive search control in mind so that the user gets maximum assistance when making difficult search control decisions.

The design of a theorem prover interface depends on the intended user. Novices need some way to define the conjecture, to view the proof and to provide proof guidance. More experienced users may also require ways to define new theories, to browse through libraries of conjectures, definitions, lemmas, *etc.*, and to switch between one part of a proof attempt and another. System developers want access to the underlying system and want to interleave testing the prover and modifying

it. Novices want a simple interface with limited functionality, so that they do not become confused and/or issue instructions at variance with their intentions. Experts want multiple views onto the prover and proof process and want a rich functionality.

User interfaces to theorem provers have exploited many interface design techniques advocated by the human computer interaction (HCI) community. These include: simple command line interfaces; via text editors (including structure editors); and graphical user interfaces (GUIs) with multiple windows, menus and icons. Each approach has its advantages and disadvantages for different groups of users. For instance, GUIs and structure editors are particularly attractive for novices, since they provide limited functionality in a readily understood format which minimises the memory requirements on the user. Experts, on the other hand, often require a richer functionality and require access to a command line interface. For convenience, this is often called from within a text editor, which facilitates recording, cutting and pasting of interactions. Many interfaces provide a combination of these techniques, so that users have access both to multiple graphical views and memory aids, on the one hand, and to rich functionality and the innards of the prover, on the other. This may enable one interface design to satisfy several different kinds of user.

The design of user interfaces to theorem provers provides a tough challenge to HCI. To guide the prover effectively requires a good understanding of the current state of the proof and the reasons for previous failures. Mathematics is inherently difficult and proofs can be very complex and subtle. Some potential users (*e.g.* systems designers using formal methods) may not be familiar with formal proof. A good interface must: assist users to understand the current proof attempt; provide mechanisms for them to interact with the proof process; avoid bewildering them with too much information, while providing what is required; and help them explore their options without imposing too high a cognitive load. This problem is by no means solved. Research on the various approaches to it can be found in the proceedings of the Workshops on User Interfaces for Theorem Provers.

11. Inductive Theorem Provers

Many theorem provers have been built with some kind of inductive capability. In this brief survey we restrict our attention to those explicit induction provers used as vehicles for significant advances in the automation of inductive reasoning. Interactive systems were briefly discussed in Section 10, page 898. Implicit induction provers are dealt with in the chapter “Inductionless induction” by Hubert Comon in this book.

11.1. The Boyer/Moore Theorem Prover

Nqthm, better known as the Boyer/Moore theorem prover, was the first theorem prover to focus specifically on the problems of search in inductive proof. It has a long history starting at the University of Edinburgh in the early 70s, [Boyer and

Moore 1973], and undergoing development at SRI International, Xerox PARC and the University of Texas at Austin, before becoming the main development system of the company, Computational Logic Inc (CLInc), founded by Boyer and Moore. It has been the subject of two books, [Boyer and Moore 1979, Boyer and Moore 1988a] and has recently been completely re-implemented as ACL2, [Kaufmann and Moore 1997, Brock, Kaufmann and Moore 1996]. There is also a version, PC-Nqthm, [Kaufmann 1988], with improved interactive facilities.

It has been applied to a massive number of conjectures — its standard corpus now stands at 24 megabytes — including some very hard problems like the verification of complete microprocessors and the proof of Gödel's incompleteness theorem. During most of its history it has been regarded as the state of the art inductive prover. More details can be found on the following web pages:

<ftp://ftp.cs.utexas.edu/pub/boyer/nqthm/index.html>

<ftp://ftp.cs.utexas.edu/pub/boyer/nqthm/nqthm-bibliography.html>

<http://www.cs.utexas.edu/users/moore/acl2/>

Both Nqthm and ACL2 use a simple, sub-first-order, type-less logic, based on Goodstein's primitive recursive arithmetic adapted from numbers to lists. Variables are regarded as implicitly universally quantified, so there is no existential quantification. There are no explicit types in the language but implicit types can be imposed either by adding conditions to conjectures or by using coercion functions which limit expressions to an appropriate range. An example of a coercion function is *num*, which makes any term into a natural number, *i.e.*

$$\text{num}(x) = \begin{cases} x & \text{if } x : \text{nat} \\ 0 & \text{otherwise} \end{cases}$$

Many of the proof techniques described above were invented by Boyer and Moore and first implemented in Nqthm. These include: recursion analysis; destructor elimination; generalisation of subterms; the flexible use of decision procedures; and the productive use of failure to decide when to apply induction. Most of these are described in [Boyer and Moore 1979, Boyer and Moore 1988a, Boyer and Moore 1988b].

11.2. RRL

The RRL (Rewrite Rule Laboratory) system was initially developed in the early 80s by Kapur, Sivakumar and Zhang at General Electric and Rensselaer Polytechnic, [Kapur, Sivakumar and Zhang 1986]. Following the move of Kapur to SUNY at Albany, the main development moved there, [Kapur and Zhang 1995]. Initially RRL used only implicit induction techniques, but subsequently it also included explicit induction, to which it made significant advances, justifying its inclusion in this survey. It has been used for the proof of some significant mathematical theorems including the Chinese remainder theorem and Ramsey's theorem.

RRL, as its name implies, is based exclusively on rewriting with, possibly conditional, equations. This is not as limiting as it first appears since any predicate can

be encoded as an equation by making the boolean truth values into terms. Indeed, RRL has competed against resolution theorem provers by translating resolution and paramodulation into forms of conditional rewriting.

One of RRLs' main contributions has been to adapt the techniques of implicit induction to explicit induction, using a technique called cover-sets, [Zhang, Kapur and Krishnamoorthy 1988]. This constructs induction rules whose well-founded relation is based on syntactic orderings developed for orienting rewrite rules, *e.g.* recursive path orderings. RRL also uses Knuth-Bendix completion for improving the computational power of the set of rewrite rules provided. More recently, it has been used as a vehicle to develop ideas about lemma discovery, [Kapur and Subramaniam 1996].

11.3. INKA

The INKA prover was initially developed in the 80s by a team of four researchers: Biundo, Hummel, Hutter and Walther, from the University of Karlsruhe, [Biundo, Hummel, Hutter and Walther 1986]. When this team broke up separate development continued at Darmstadt by Walther and Saarbrücken by Hutter, [Hutter and Sengler 1996]. INKA is based on a resolution theorem prover for clausal, first-order logic. At various times in its history it has formed the inductive component of larger provers, *e.g.* the MarkGraf Karl prover, [Eisinger, Siekmann, Smolka, Unvericht and Walther 1980], the Ω mega prover, [Benzmüller, Cheikhrouhou, Fehrer, Fiedler, Huang, Kerber, Kohlhase, Meirer, Melis, Schaarschmidt, Siekmann and Sorge 1997], and the VSE system, [Hutter, Langenstein, Siekmann and Stephan 1996]. It has been used for the verification of software of industrial interest and significant size. More details can be found on the following web page:

<http://www.dfki.de/vse/systems/inka/>

Each of the INKA authors has made significant contributions to the proof techniques described above. Walther's contributions have been the proof of termination of recursive functions, [Walther 1994b], and the construction of induction rules, [Walther 1992, Walther 1993]. Hutter's contributions have been in techniques for guiding search, especially the development and application of rippling, [Hutter 1990, Hutter 1997, Hutter and Kohlhase 1997]. Hummel's contribution was the development of heuristics for generalisation, [Hummel 1990]. Biundo's contribution was the synthesis of programs by the proof of existential theorems, [Biundo 1988].

11.4. *Oyster/CIAM*

The *Oyster/CIAM* was developed at the University of Edinburgh in the 90s by a large team led by the author, [Bundy, van Harmelen, Horn and Smaill 1990]. *Oyster* is a Prolog re-implementation by Horn of NUPRL, *i.e.* it is a tactic-based proof editor based on Martin-Löf constructive type theory. *CIAM* is a proof planner which guides *Oyster*. The behaviour of each *Oyster* tactic is specified in a meta-

language. *CIAM* reasons in this meta-language to construct a customised tactic for each conjecture and then supplies this tactic to *Oyster*. These tactics include rippling. The combined system has been used for the verification of a complete microprocessor and the synthesis of the rippling tactic. More details can be found on the web page:

<http://dream.dai.ed.ac.uk/home.html>

The contributions of the Edinburgh team include: proof planning, [Bundy 1988, Bundy, van Harmelen, Hesketh and Smaill 1991, Bundy 1991]; rippling, [Bundy 1988, Bundy et al. 1993]; recursion analysis and ripple analysis, [Stevens 1988, Bundy, van Harmelen, Hesketh, Smaill and Stevens 1989];, and the productive use of failure including techniques for choosing induction rules, speculating lemmas and generalising conjectures, [Ireland 1992, Ireland and Bundy 1996b, Ireland and Bundy 1996a]. Proof planning has also been adapted to lift the level of interaction and implemented in the semi-automated prover, *Barnacle*, [Lowe and Duncan 1997, Lowe, Bundy and McLean 1995].

12. Conclusion

In this chapter we have surveyed the automation of inductive inference. We have seen that automating induction is necessary for some of the most important applications of automated reasoning, in particular, meeting the proof obligations that arise from formal methods of system development. But we have also seen that inductive proof raises difficult search control problems for automation. The construction of appropriate induction rules, the use of intermediate lemmas and the generalisation of conjectures all introduce infinite branch points into the search space.

It is necessary to develop special search control techniques to solve these problems. Since they are undecidable problems, these search control techniques are necessarily partial and heuristic, *i.e.* they will sometimes fail and are always open to improvement. We can hope only that they help prove a significant proportion of the inductive theorems that arise in practice. Sometimes the failure of a particular technique can be analysed to suggest what additional techniques should be applied to patch the initial proof attempt.

There has been significant progress over the last three decades of research. Some quite subtle and long proofs can be found automatically. Unfortunately, automated inductive theorem proving is not yet robust enough to be used unaided and reliably on problems of industrial interest. For practical inductive theorem proving it is currently necessary to use an interactive system where the user provides guidance to the proof at critical stages. However, automation is a vital adjunct to interactive proof to reduce the burden on the user so that proofs can be completed within a reasonable timescale.

The following are some of the key research issues for future research in inductive theorem proving.

1. Practical proof problems do not consist of induction alone. It is vital to integrate inductive techniques with non-inductive proof techniques, in particular,

successful techniques like model checking, decision procedures, rewriting, built-in unification, *etc.* Much progress has already been made in this area by systems in everyday use, but more is needed.

2. In semi-automated systems it is sometimes difficult for users to orient themselves within a failed automatic proof attempt to suggest an appropriate patch. More automatic analysis of the failed attempt is required to put the user in context and suggest what kinds of interaction might be most effective.
3. The heuristics for lemma speculation, generalisation and induction rule choice are always in need of improvement. The first two are especially weak at present.
4. Most work on automation has focussed on the universal fragment of first-order logic, but many practical problems are not naturally formulated within this fragment. More work is needed to extend existing heuristics to deal with existential quantification and higher-order logic.

For a longer introduction to automated inductive theorem proving the reader is recommended to read, [Walther 1994a].

Acknowledgments

I am grateful to Alessandro Armando, Richard Boulton, Jeremy Gow, Ian Green, Andrew Ireland, Mike Jackson, Helen Lowe, Dave McAllester, Andrei Voronkov, Toby Walsh and Christoph Walther for feedback on an earlier draft. The author's research in this area is supported by EPSRC grants GR/M/45030 and GR/L/14381.

Bibliography

- AUBIN R. [1976], Mechanizing Structural Induction, PhD thesis, University of Edinburgh.
- BASIN D. A. AND WALSH T. [1993], Difference unification, *in* R. Bajcsy, ed., 'Proc. 13th Intern. Joint Conference on Artificial Intelligence (IJCAI '93)', Vol. 1, Morgan Kaufmann, San Mateo, CA, pp. 116–122. Also available as Technical Report MPI-I-92-247, Max-Planck-Institut für Informatik.
- BASIN D. AND WALSH T. [1996], 'A calculus for and termination of rippling', *Journal of Automated Reasoning* 16(1–2), 147–180.
- BENZMÜLLER C., CHEIKHROUHO L., FEHRER D., FIEDLER A., HUANG X., KERBER M., KOHLHASE K., MEIRER A., MELIS E., SCHAARSCHMIDT W., SIEKMANN J. AND SORGE V. [1997], *Omega: Towards a mathematical assistant*, *in* W. McCune, ed., '14th International Conference on Automated Deduction', Springer-Verlag, pp. 252–255.
- BIUNDO S. [1988], Automated synthesis of recursive algorithms as a theorem proving tool, *in* Y. Kodratoff, ed., 'Eighth European Conference on Artificial Intelligence', Pitman, pp. 553–8.
- BIUNDO S., HUMMEL B., HUTTER D. AND WALTHER C. [1986], The Karlsruhe induction theorem proving system, *in* J. Siekmann, ed., '8th International Conference on Automated Deduction', Springer-Verlag, pp. 672–674. Springer Lecture Notes in Computer Science No. 230.
- BOYER R. S. AND MOORE J. S. [1973], Proving theorems about LISP functions, *in* N. Nilsson, ed., 'Proceedings of the third IJCAI', International Joint Conference on Artificial Intelligence, pp. 486–493. Also available from Edinburgh as DCL memo No. 60.
- BOYER R. S. AND MOORE J. S. [1979], *A Computational Logic*, Academic Press. ACM monograph series.

- BOYER R. S. AND MOORE J. S. [1988a], *A Computational Logic Handbook*, Academic Press. Perspectives in Computing, Vol 23.
- BOYER R. S. AND MOORE J. S. [1988b], Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic, in J. E. Hayes, J. Richards and D. Michie, eds, 'Machine Intelligence 11', pp. 83–124.
- BROCK B., KAUFMANN M. AND MOORE J. S. [1996], Acl2 theorems about commercial micro-processors, in M. Srivas and A. Camilleri, eds, 'Formal Methods in Computer-Aided Design (FMCAD'96)', Springer-Verlag, pp. 275–293.
- BRYANT R. E. [1992], 'Symbolic boolean manipulation with ordered binary-decision diagrams', *ACM Computing Surveys* 24(3), 293–318.
- BUNDY A. [1988], The use of explicit plans to guide inductive proofs, in R. Lusk and R. Overbeek, eds, '9th International Conference on Automated Deduction', Springer-Verlag, pp. 111–120. Longer version available from Edinburgh as DAI Research Paper No. 349.
- BUNDY A. [1991], A science of reasoning, in J.-L. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of Alan Robinson', MIT Press, pp. 178–198. Also available from Edinburgh as DAI Research Paper 445.
- BUNDY A., STEVENS A., VAN HARMELEN F., IRELAND A. AND SMAILL A. [1993], 'Rippling: A heuristic for guiding inductive proofs', *Artificial Intelligence* 62, 185–253. Also available from Edinburgh as DAI Research Paper No. 567.
- BUNDY A., VAN HARMELEN F., HESKETH J. AND SMAILL A. [1991], 'Experiments with proof plans for induction', *Journal of Automated Reasoning* 7, 303–324. Earlier version available from Edinburgh as DAI Research Paper No 413.
- BUNDY A., VAN HARMELEN F., HESKETH J., SMAILL A. AND STEVENS A. [1989], A rational reconstruction and extension of recursion analysis, in N. S. Sridharan, ed., 'Proceedings of the Eleventh International Joint Conference on Artificial Intelligence', Morgan Kaufmann, pp. 359–365. Also available from Edinburgh as DAI Research Paper 419.
- BUNDY A., VAN HARMELEN F., HORN C. AND SMAILL A. [1990], The Oyster-Clam system, in M. E. Stickel, ed., '10th International Conference on Automated Deduction', Springer-Verlag, pp. 647–648. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.
- BUNDY A., VAN HARMELEN F., SMAILL A. AND IRELAND A. [1990], Extensions to the rippling-out tactic for guiding inductive proofs, in M. E. Stickel, ed., '10th International Conference on Automated Deduction', Springer-Verlag, pp. 132–146. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 459.
- CLARKE E. AND SCHLINGLOFF H. [2001], Model checking, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 24, pp. 1635–1790.
- CONSTABLE R. L., ALLEN S. F., BROMLEY H. M. ET AL. [1986], *Implementing Mathematics with the Nuprl Proof Development System*, Prentice Hall.
- DARLINGTON J. [1978], 'A synthesis of several sorting algorithms', *Acta Informatica* 11, 1–30.
- EISINGER N., SIEKMANN J., SMOLKA G., UNVERICHT E. AND WALTHER C. [1980], The MarkGraf Karl Refutation Procedure, in S. Hardy, ed., 'Proceedings of AISB-80', Society for the Study of Artificial Intelligence and Simulation of Behaviour.
- FERMÜLLER C., LEITSCH A., HUSTADT U. AND TAMMET T. [2001], Resolution decision procedures, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 25, pp. 1791–1849.
- FRANOVA M. AND KODRATOFF Y. [1992], Predicate synthesis from formal specifications, in 'Proceedings of ECAI-92', European Conference on Artificial Intelligence, pp. 87–91.
- GENTZEN G. [1969], *The Collected Papers of Gerhard Gentzen*, North Holland. Edited by Szabo, M. E.
- GÖDEL K. [1931], 'Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i', *Monatsh. Math. Phys.* 38, 173–198. English translation in [Heijenoort 1967].

- GORDON M. J., MILNER A. J. AND WADSWORTH C. P. [1979], *Edinburgh LCF - A mechanised logic of computation*, Vol. 78 of *Lecture Notes in Computer Science*, Springer-Verlag.
- HEIJENOORT J. v. [1967], *From Frege to Gödel: a source book in Mathematical Logic, 1879-1931*, Harvard University Press, Cambridge, Mass.
- HESKETH J. T. [1991], *Using Middle-Out Reasoning to Guide Inductive Theorem Proving*, PhD thesis, University of Edinburgh.
- HOWARD W. A. [1980], The formulae-as-types notion of construction, in J. P. Seldin and J. R. Hindley, eds, 'To H. B. Curry; Essays on Combinatory Logic, Lambda Calculus and Formalism', Academic Press, pp. 479-490.
- HUMMEL B. [1990], *Generation of induction axioms and generalisation*, PhD thesis, Universität Karlsruhe.
- HUTTER D. [1990], Guiding inductive proofs, in M. E. Stickel, ed., '10th International Conference on Automated Deduction', Springer-Verlag, pp. 147-161. *Lecture Notes in Artificial Intelligence* No. 449.
- HUTTER D. [1997], 'Coloring terms to control equational reasoning', *Journal of Automated Reasoning* 18(3), 399-442.
- HUTTER D. AND KOHLHASE M. [1997], A colored version of the λ -Calculus, in W. McCune, ed., '14th International Conference on Automated Deduction', Springer-Verlag, pp. 291-305. Also available as SEKI-Report SR-95-05.
- HUTTER D., LANGENSTEIN, B. SENGLER C., SIEKMANN J. AND STEPHAN W. [1996], Deduction in the verification support environment, in 'Formal Methods Europe '96', Springer-Verlag. Springer Lecture Notes No. 1051.
- HUTTER D. AND SENGLER C. [1996], INKA: the next generation, in M. A. McRobbie and J. K. Slaney, eds, '13th International Conference on Automated Deduction', Springer-Verlag, pp. 288-292. Springer Lecture Notes in Artificial Intelligence No. 1104.
- IRELAND A. [1992], The Use of Planning Critics in Mechanizing Inductive Proofs, in A. Voronkov, ed., 'International Conference on Logic Programming and Automated Reasoning - LPAR 92, St. Petersburg', Lecture Notes in Artificial Intelligence No. 624, Springer-Verlag, pp. 178-189. Also available from Edinburgh as DAI Research Paper 592.
- IRELAND A. AND BUNDY A. [1996a], Extensions to a Generalization Critic for Inductive Proof, in M. A. McRobbie and J. K. Slaney, eds, '13th International Conference on Automated Deduction', Springer-Verlag, pp. 47-61. Springer Lecture Notes in Artificial Intelligence No. 1104. Also available from Edinburgh as DAI Research Paper 786.
- IRELAND A. AND BUNDY A. [1996b], 'Productive use of failure in inductive proof', *Journal of Automated Reasoning* 16(1-2), 79-111. Also available as DAI Research Paper No 716, Dept. of Artificial Intelligence, Edinburgh.
- KAPUR D., SIVAKUMAR G. AND ZHANG H. [1986], RRL: A rewrite rule laboratory, in J. Siekmann, ed., '8th International Conference on Automated Deduction', Springer-Verlag, pp. 692-693.
- KAPUR D. AND SUBRAMANIAM M. [1996], Lemma discovery in automating induction, in M. A. McRobbie and J. K. Slaney, eds, '13th International Conference on Automated Deduction (CADE-13)', CADE, Springer, pp. 538-552.
- KAPUR D. AND ZHANG H. [1995], 'An overview of rewrite rule laboratory (RRL)', *J. of Computer Mathematics with Applications* 29(2), 91-114.
- KAUFMANN M. [1988], A user's manual for an interactive enhancement to the Boyer-Moore theorem prover, Technical Report 19, Computational Logic Inc.
- KAUFMANN M. AND MOORE J. S. [1997], 'An industrial strength theorem prover for a logic based on common lisp', *IEEE Transactions on Software Engineering* 23(4), 203-213.
- KIRBY L. A. AND PARIS J. [1982], 'Accessible independence results for Peano Arithmetic', *Bull. London Math. Soc.* 14, 285-293.
- KOLBE T. AND WALTHER C. [1998], Proof analysis, generalization and reuse, in W. Bibel and P. H. Schmitt, eds, 'Automated Deduction - A Basis for Applications, Vol. II Systems and

- Implementation Techniques', Applied Logic Series, vol. 9, Kluwer Academic Publishers, Dordrecht, Boston, London, pp. 189-219.
- KRAAN I., BASIN D. AND BUNDY A. [1996], 'Middle-out reasoning for synthesis and induction', *Journal of Automated Reasoning* 16(1-2), 113-145. Also available from Edinburgh as DAI Research Paper 729.
- KREISEL G. [1965], Mathematical logic, in T. Saaty, ed., 'Lectures on Modern Mathematics', Vol. 3, J. Wiley & Sons, pp. 95-195.
- LOWE H., BUNDY A. AND MCLEAN D. [1995], The use of proof planning for co-operative theorem proving, Research Paper 745, Dept. of Artificial Intelligence, University of Edinburgh. Appeared in the special issue of the Journal of Symbolic Computation on graphical user interfaces and protocols, February 1998.
- LOWE H. AND DUNCAN D. [1997], XBarnacle: Making theorem provers more accessible, in W. McCune, ed., '14th International Conference on Automated Deduction', Springer-Verlag, pp. 404-408.
- MARTIN-LÖF P. [1979], Constructive mathematics and computer programming, in '6th International Congress for Logic, Methodology and Philosophy of Science', Hanover, pp. 153-175. Published by North Holland, Amsterdam. 1982.
- MONROY R., BUNDY A. AND IRELAND A. [1994], Proof Plans for the Correction of False Conjectures, in F. Pfenning, ed., '5th International Conference on Logic Programming and Automated Reasoning, LPAR'94', Lecture Notes in Artificial Intelligence, v. 822, Springer-Verlag, Kiev, Ukraine, pp. 54-68. Also available from Edinburgh as DAI Research Paper 681.
- MOORE J. S. [1974], Computational Logic: Structure sharing and proof of program properties, part II, PhD thesis, University of Edinburgh. Available from Edinburgh as DCL memo no. 68 and from Xerox PARC, Palo Alto as CSL 75-2.
- NELSON G. AND OPPEN D. C. [1979], 'Simplification by cooperating decision procedures', *ACM Transactions on Programming Languages and Systems* 1(2), 245-257.
- NELSON G. AND OPPEN D. C. [1980], 'Fast decision procedures based on congruence closure', *Journal of the ACM* 27(2), 356-364. Also: Stanford CS Report STAN-CS-77-646, 1977.
- PRESBURGER M. [1930], Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt, in 'Sprawozdanie z I Kongresu matematyków słowiańskich, Warszawa 1929', Warsaw, pp. 92-101, 395. Annotated English version also available [Stansifer 1984].
- PROTZEN M. [1992], Disproving conjectures, in D. Kapur, ed., '11th International Conference on Automated Deduction', Saratoga Springs, NY, USA, pp. 340-354. Published as Springer Lecture Notes in Artificial Intelligence, No 607.
- PROTZEN M. [1994], Lazy generation of induction hypotheses, in A. Bundy, ed., '12th International Conference on Automated Deduction', Lecture Notes in Artificial Intelligence, Vol. 814, Springer-Verlag, Nancy, France, pp. 42-56.
- SEGLER C. [1996], Termination of algorithms over non-freely generated datatypes, in M. McRobbie and J. Slaney, eds, '13th International Conference on Automated Deduction', LNAI 1104, Springer Verlag, pp. 121-135.
- SHOSTAK R. E. [1984], 'Deciding combinations of theories', *Journal of the ACM* 31(1), 1-12.
- STANSIFER R. [1984], Presburger's article on integer arithmetic: Remarks and translation, Technical Report TR 84-639, Department of Computer Science, Cornell University.
- STEVENS A. [1988], A rational reconstruction of Boyer & Moore's technique for constructing induction formulas, in Y. Kodratoff, ed., 'The Proceedings of ECAI-88', European Conference on Artificial Intelligence, pp. 565-570. Also available from Edinburgh as DAI Research Paper No. 360.
- TURING A. M. [1936-7], 'On computable numbers, with an application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society* (2) 42, 230-265.

- WALTHER C. [1992], Computing induction axioms, *in* A. Voronkov, ed., 'International Conference on Logic Programming and Automated Reasoning – LPAR 92, St. Petersburg', Lecture Notes in Artificial Intelligence No. 624, Springer-Verlag.
- WALTHER C. [1993], Combining induction axioms by machine, *in* 'Proceedings of IJCAI-93', International Joint Conference on Artificial Intelligence, pp. 95–101.
- WALTHER C. [1994a], Mathematical induction, *in* C. J. H. D. M. Gabbay and J. A. Robinson, eds, 'Handbook of Logic in Artificial Intelligence and Logic Programming', Vol. 2, Oxford University Press, Oxford, pp. 122–227.
- WALTHER C. [1994b], 'On proving termination of algorithms by machine', *Artificial Intelligence* 71(1), 101–157.
- YOSHIDA T., BUNDY A., GREEN I., WALSH T. AND BASIN D. [1994], Coloured rippling: An extension of a theorem proving heuristic, *in* A. G. Cohn, ed., 'Proceedings of ECAI-94', John Wiley, pp. 85–89.
- ZHANG H., KAPUR D. AND KRISHNAMOOTHY M. S. [1988], A mechanizable induction principle for equational specifications, *in* R. Lusk and R. Overbeek, eds, '9th International Conference on Automated Deduction', Springer-Verlag, pp. 162–181.

Main Index

Symbols

Ω prover	902
λ calculus	896

A

ACL2	901
------------	-----

B

Barnacle	903
base case	847
binary trees	851
Boyer/Moore theorem prover	900

C

Clam	902
coercion functions	901
conflict rule of unification	885
constructive type theory	896
constructor function	860
constructor functions	851
constructor-style	849
constructor-style proofs	860
containment	867
containment formula	868
contains	867
cover-sets	902
cross fertilization	859
Curry-Howard isomorphism	896
cut formula	864

D

destructor elimination	860
destructor function	860
destructor-style	849
destructor-style proofs	860
difference matching	885
difference unification	882, 885

E

explicit induction	848
--------------------------	-----

F

fertilization	855
flawed induction variable candidates ..	867
free datatypes	851

G

generalise apart	872
ground difference unification	885

I

implicit induction	848
induction	847
induction conclusion	847
induction hypothesis	847
induction term	847
induction variable	847
inductionless induction	848
inductive completion	848
inductive learning	847
INKA prover	902

K

Knuth-Bendix completion	848, 902
-------------------------------	----------

L

linear arithmetic	863
lists	851
longitudinal wave-rules	878

M

MarkGraf Karl prover	902
mathematical induction	847

N

narrowing	891
natural numbers	851
non-free datatypes	852
Nqthm	900
NUPRL	896
Noetherian induction	849

O

Oyster	902
--------------	-----

P

PC-Nqthm	901
Peano induction	847
philosophical induction	847
polarity	850, 857
Presburger arithmetic	863
proofs as programs	896

R

r-descriptions	868
recursion analysis	865
recursive arguments	856
recursive datatypes	851
recursive definitions	851
redex	856

replacement axioms	857
rewrite rule of inference	855
rewrite rules	855
ripple analysis	887
ripple-out	876
Rippling	876
rippling-in	878
rippling-sideways	878
RRL	901

S

S-expressions	851
second-order unification	891
separated union	868
sinks	878
skeleton	876
step case	847
strong fertilization	859
structural recursion	852
subsumes	867

T

tacticals	899
tactics	899
terminating	852
termination	862
the productive use of failure	890
transverse wave-rules	878

U

unflawed induction variable candidates	866
--	-----

V

VSE system	902
------------------	-----

W

wave-fronts	876
wave-holes	876
wave-rules	877
weak fertilization	859
weakening	883
well-defined	853
well-founded induction	849

Name Index

A

Aubin876

B

Basin884

Biundo896, 902

Boyer859, 863, 869, 901

Bundy876, 902

F

Franova875

G

Gödel864

Gentzen864

Goodstein864, 901

Green903

H

Horn902

Hummel902

Hutter876, 902

I

Ireland890, 903

K

Kapur901

Kirby864

Kodratoff875

Kraan896

Kreisel864

L

Lowe903

M

Martin-Löf896

Milner899

Monroy875

Moore859, 863, 869, 901

P

Paris864

Peano864

Presburger862

Protzen875, 898

S

Sivakumar901

Smaill903

Stevens867, 869, 903

T

Turing864

V

van Harmelen903

W

Walsh884

Walther865, 869, 898, 902

Z

Zhang901

Inductionless Induction

Hubert Comon

SECOND READERS: Bernhard Gramlich, David McAllester, and
Donald Sannella.

Contents

1	Introduction	915
1.1	Inductive theorem proving	915
1.2	A few words explaining the title	916
1.3	Inductionless induction versus implicit induction	917
1.4	Outline of the chapter	918
2	Formal background	919
2.1	Terms and clauses	919
2.2	Equational deduction	920
2.3	Inductive theory	920
2.4	Constructors and sufficient completeness	921
2.5	Term Rewriting	922
2.6	Standard Completion	923
3	General Setting of the Inductionless Induction Method	925
4	Inductive completion methods	927
4.1	Redundancy and Saturation	927
4.2	Inductive saturation strategies	929
4.3	Ordered strategies	930
4.4	Normal axiomatizations	933
4.5	Examples	934
4.6	Further refinements	936
5	Examples of Axiomatizations \mathcal{A} from the Literature	938
5.1	Musser's approach [Musser 1980]	938
5.2	Huet and Hullot's method [Huet and Hullot 1982]	941
5.3	Jouannaud and Kounalis approach [Jouannaud and Kounalis 1989]	943
5.4	Bachmair's approach [Bachmair 1988]	946
5.5	Further examples of normal axiomatizations	947
6	Ground Reducibility	948
6.1	Automata techniques	949
6.2	Test sets	952
6.3	Ground normal form languages	954
6.4	Sufficient Completeness	956
6.5	Limitations	957

HANDBOOK OF AUTOMATED REASONING

Edited by Alan Robinson and Andrei Voronkov

© 2001 Elsevier Science Publishers B.V. All rights reserved

7	A comparison between inductive proofs and proofs by consistency	957
	Bibliography	958
	Index	961

1. Introduction

1.1. Inductive theorem proving

The aim of inductive theorem proving is to prove statements about particular structures, e.g. natural numbers, lists, arrays... Natural first order axiomatizations of such familiar objects fail to capture the true statements we want to prove.

1.1. EXAMPLE. Assume that we give the following simple definition of $+$ on the natural numbers (in base 1):

$$E = \begin{cases} 0 + x & = & x \\ s(x) + y & = & s(x + y) \end{cases}$$

$x + 0 = x$ is not a logical consequence of the above definition of $+$, as there are models of E which do not satisfy the equation $x + 0 = x$. For instance, consider the interpretation I whose domain is the two elements set $\{0, a\}$ and such that s_I , the interpretation of s in I , is the identity $s_I(0) = 0$, $s_I(a) = a$ and $+_I$, the interpretation of $+$ in I , is the second projection: $u +_I v$ is v . Then the equations of E are satisfied in I . However $x + 0 = x$ is not valid in I , since $a + 0$ equals 0 in I .

The reason of the failure is that our domain of interpretation contains some element (namely a) which cannot be reached from 0 using a number of successors. When writing our specification, our intention was to define the addition on *natural numbers* i.e. terms which are built from 0 using a number of successors. Such a structure will be called *the structure freely generated by* $\{0, s\}$, or (more generally) *the minimal Herbrand model* of the axioms.

If we use the knowledge that we only consider this particular structure, it is not difficult to prove $x + 0 = x$. For instance, we may use our familiar Peano's induction scheme: $0 + 0 = 0$ by the first equation and, if $s^n(0) + 0 = s^n(0)$, then $s^{n+1}(0) + 0 = s(s^n(0) + 0)$ by the second equation and $s(s^n(0) + 0) = s^{n+1}(0)$ by induction hypothesis. Note however that Peano's induction scheme is most naturally expressed in second-order logic.

More generally, most structures of interest in computer science can be defined using first-order terms. For instance, natural numbers can be modeled by $s(s(\dots s(0)))$ and lists can be modeled by $cons(a_1, cons(a_2, \dots cons(a_n, nil)))$. The objects in the structure are represented as terms built over a given signature (the *constructors*), or, more generally, as classes of terms modulo some congruence relation.

1.2. EXAMPLE. Instead of natural numbers as before, we consider a specification of the group of integers; they are represented using successor as before and the negation sign, as usual:

$$(1) \quad \left\{ \begin{array}{rcl} -0 & = & 0 \\ - - x & = & x \\ s(-s(x)) & = & -x \\ 0 + x & = & x \\ s(x) + y & = & s(x + y) \end{array} \right.$$

The main difference now is that the three first equations specify some relations between the negation sign, 0 and s . Hence the intended structure is no longer freely generated: it is the quotient of the set of terms built on $0, -, s, +$ by the congruence generated by the above set of equations: this is again the *minimal Herbrand model* of the set of axioms. If our specification is “complete” with respect to what we intended, then a set of representatives will consist of $\{s^n(0) \mid n \in \mathbb{N}\} \cup \{-s^n(0) \mid n \in \mathbb{N}\}$. The commutativity $x + y = y + x$ is an inductive consequence of (1). But it is not completely straightforward to see how to prove it.

The question is: how can we (automatically) prove some properties in such minimal Herbrand structures? One possible approach is to use *induction* as we did in our first example above. Mechanizing induction is precisely the purpose of Chapter 13 of this Handbook [Bundy 2001]. The purpose of the present chapter is to explain a method which achieves the same goal, but without the use of an induction scheme. Let us first review and compare the different methods.

1.2. A few words explaining the title

The expression “inductionless induction” was coined by D. Lankford [Lankford 1981]. It refers to a proof technique in minimal Herbrand models which does not make use of explicit induction rules (hence differs from the inductive proof methods described in [Bundy 2001] (Chapter 13 of this Handbook): we will stay within classical first-order logic. Inductionless induction should rather be called *proof by consistency*, following D. Kapur and D. Musser’s terminology [Kapur and Musser 1987], as we will see. This method was discovered in the years 80-82 by several authors [Musser 1980, Goguen 1980, Lankford 1981, Huet and Hullot 1982] and a lot of work has been devoted to it since, mainly in the term rewriting community. This chapter relies on this work. It aims at giving a general presentation, encompassing most of the results in the field.

The inductionless induction approach is the basis of several inductive theorem provers. In general such theorem provers are small: nothing comparable with the Boyer and Moore theorem prover [Boyer and Moore 1979] in terms of applications. Probably the most well-known one is part of the RRL system [Kapur and Zhang 1995] (this system also includes a cover set induction tool which is not a proof by consistency technique, see below). There are some other implementations such

as ReDuX¹ or UNICOM [Gramlich 1990c, Gramlich and Lindner 1991]. However, the main advantage of inductionless induction is the ability to use general purpose first-order theorem provers for inductive theorem proving; there is no real need to develop a dedicated tool. The only requirement is to design dedicated strategies within a first-order theorem prover.

The core of the method for the purely equational case is explained in detail in L. Bachmair's book [Bachmair 1991]. The non-equational case is studied in [Comon and Nieuwenhuis 1998].

Basically, inductionless induction is a “proof by consistency” method. Assume that E is a set of axioms (e.g. the two equations defining $+$ in example 1.1) and that there is (up to isomorphism) a single minimal Herbrand model \mathcal{H} of E (e.g. \mathbb{N} , in example 1.1). The idea is to use a (first-order) axiomatization \mathcal{A} of \mathcal{H} such that $\phi \cup \mathcal{A} \cup E$ is consistent if and only if ϕ is valid in \mathcal{H} . In example 1.1, such an axiomatization \mathcal{A} can be $\{\forall x. 0 \neq s(x); \forall x, y. s(x) = s(y) \Rightarrow x = y\}$, as we will see. If we want to prove the conjecture $x + y = y + x$. We add it to $E \cup \mathcal{A}$ and try to derive an inconsistency.

Then, we rely on *saturation techniques* which are introduced in [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook) and will be recalled here. In the example, it is possible to saturate $E \cup \mathcal{A} \cup \phi$ without deriving an inconsistency: this proves that commutativity is indeed valid in \mathbb{N} . This example is fully developed in example 4.13.

1.3. Inductionless induction versus implicit induction

We chose to restrict ourselves to proofs by consistency, i.e. we do not cover methods using an induction scheme. There are however several techniques which use the ideas developed in the area of proofs by consistency and which also aim at proving inductive theorems. Let us cite for instance the cover set method [Zhang 1988], the test set method [Bouhoula and Rusinowitch 1995] and the rewriting induction method [Reddy 1990]. All these methods are roughly *implicit induction* methods: the induction scheme is not necessarily known beforehand. More precisely, each of these methods provides a set of terms (or a set of pairs (context, term)) which is used for the replacement of the induction variable (depending on the context). Such replacements produce new conjectures which can be simplified by strictly smaller instances of the original conjecture (the induction hypothesis). The proof is completed when all newly generated conjectures are simplified into known inductive theorems.

¹see <http://www-sr.informatik.uni-tuebingen.de/~buendgen/redux.html>

1.3. **EXAMPLE.** Consider the following (other) definition of the addition on integers:

$$\left\{ \begin{array}{lcl} 0 + x & = & x \\ s(x) + y & = & s(x + y) \\ s(p(x)) & = & x \\ p(s(x)) & = & x \\ p(x) + y & = & p(x + y) \end{array} \right.$$

All these equations can be used from left to right, yielding a terminating system. Consider the conjecture $x + 0 = x$. The cover set method will overlap the conjecture with the left members, which yields to three more conjectures: $0 + 0 = 0$, $s(x) + 0 = s(x)$, $p(x) + 0 = p(x)$ (in other words we replaced x with all elements of the *cover set* $\{0, s(x), p(x)\}$). The new conjectures can be simplified with the equations defining $+$ and with the induction hypothesis $x + 0 = x$ in which x is frozen (cannot be instantiated). In our example, $0 + 0$ reduces to 0 using $0 + x = x$, $s(x) + 0$ reduces to $s(x + 0)$ using the second equation, then to $s(x)$ using the induction hypothesis and finally $p(x + 0)$ reduces to $p(x)$ using the last equation and the induction hypothesis. This completes the proof.

The main difference between explicit induction techniques and implicit induction techniques is roughly that the former rely on a semantic ordering while the latter rely on a syntactic ordering (the one which shows the termination of the definitions). However, both use an induction scheme, which is not the case in our chapter.

One should avoid any confusion between such implicit induction techniques and the inductionless induction method. The methods are very different in nature. For instance, in principle, we can prove properties such as $x \neq s(x)$ using inductionless induction. This is not possible with implicit induction methods. The reason is that inductionless induction aims at proving properties in one particular model: the minimal Herbrand model, whereas implicit induction aims at proving properties which hold true in *all* constructor-generated models (e.g. 0 is equal to $s(0)$ in the one-element model of the axioms of example 1.1). The two notions will coincide on positive properties (see lemma 2.2).

Let us insist: one of the main features of inductionless induction is the possibility to use classical first-order theorem provers. Cover set (resp. test set, resp. rewriting) induction are very interesting and useful techniques. However, they use an extra rule, which is not valid in first-order logic and hence, they need dedicated tools. These methods are not relevant to our chapter.

1.4. *Outline of the chapter*

The subject of this chapter is to explain inductionless induction, to explain how and why it works. More precisions on the setting will be given in section 3 after recalling some basic background in section 2. Then, two main issues have to be discussed:

What are the deduction rules used in this context? The method originally relies on completion procedures (which are described in [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook). In the case of proofs by consistency, the deduction system consists in some restrictions of the Knuth-Bendix completion procedure, as explained in [Bachmair 1991, Fribourg 1989, Bachmair and Dershowitz 1994].

The Knuth-Bendix completion procedure can be extended to arbitrary clauses with equality, roughly replacing completion with *saturation* and simplification with *redundancy*, see [Bachmair and Ganzinger 2001, Nieuwenhuis and Rubio 2001] (Chapters 2 and 7 of this Handbook). Following this extension, we give in section 4 a general inductive saturation method, which is not restricted to the purely equational case.

How can we build an axiomatization \mathcal{A} which has the desired properties? Some techniques are presented in section 5. They are mainly reformulations of methods suggested in [Musser 1980, Huet and Hullot 1982, Jouannaud and Kounalis 1986, Bachmair 1988, Dershowitz 1982]. Basically, \mathcal{A} expresses that two equal terms should be *ground reducible*. This notion plays such an important role that a whole section (section 6), is devoted to it.

All these techniques have been actually developed in the framework of equational theories. They carry over to clauses with equality, with the proviso that, in general, there is no known axiomatization \mathcal{A} for which the consistency of $e \cup \mathcal{A}$ can be decided for every equation e . Nevertheless, it is possible to design a technique which may fail to prove consistency, but always succeeds in detecting inconsistencies; we get refutationally complete proof techniques (see [Ganzinger and Stuber 1992]). Of course, they may also prove sometimes the consistency of $e \cup \mathcal{A}$. However, in the clausal case, it is not only hard to prove consistency, but also to get a saturated set.

As a conclusion, we develop in section 7 a short (maybe controversial) comparison between the different methods.

2. Formal background

2.1. Terms and clauses

We will use finite *terms* constructed out of a finite set of *function symbols* F whose arity (number of arguments) is fixed and an infinite set of *variables* X . The set of terms is written $T(F, X)$. The set of variables of a term is written $\text{Vars}(t)$. A *ground term* is a term which does not contain any variable. The set of ground terms is written $T(F)$. The identity (uninterpreted equality) on terms is written \equiv . The terms are also identified, as usual, with mapping from prefix-closed sets of strings of natural numbers called *positions* to $F \cup X$. The label at a position p of t is written $t(p)$. The subterm of t at position p is denoted $t|_p$.

Substitutions are (simultaneous) replacements of variables with terms. Concerning substitutions, we use the same definitions and notations as in [Baader and

Snyder 2001] (Chapter 8 of this Handbook).

In particular, when x_1, \dots, x_n are variables, $\{x_1 \mapsto t_1; \dots; x_n \mapsto t_n\}$ maps each x_i to t_i and the other variables to themselves. A *ground substitution* maps every variable to a ground term. A term t *encompasses* a term u if there is an instance of u which is a subterm of t .

An *atomic formula* (also called a *positive literal*) is either an equation $s = t$ between two terms or a predicate symbol applied to terms $P(t_1, \dots, t_n)$. A *literal* is either an atomic formula or its negation. A (general) *clause* is a disjunction of literals. A *positive clause* is a disjunction of positive literals. A *Horn clause* is a clause which contains exactly one positive literal.

2.2. Equational deduction

If E is a set of equations between terms of $T(F, X)$, $=_E$ is the smallest congruence on $T(F, X)$ such that $s\sigma =_E t\sigma$ for all equations $s = t \in E$ and for all substitutions σ .

Given a set of equations E , $s = t$ is a *consequence* of E if $s =_E t$. Equivalently $s = t$ is a consequence of E if it can be proved using a finite sequence of *replacement of equals by equals*: $=_E$ is the reflexive and transitive closure of the binary relation $\xleftrightarrow[E]{\sigma} . u \xleftrightarrow[E]{\sigma} v$ if there is an equation $l = r \in E$ and a substitution σ such that u is obtained from v by replacing some occurrence of $l\sigma$ with $r\sigma$ (resp. replacing some occurrence of $r\sigma$ with $l\sigma$). If we want to make precise the equation and/or the substitution, we can use them as indices of the relation as in: $u \xleftrightarrow[l=r]{\sigma} v$.

2.3. Inductive theory

A *Herbrand interpretation* of a set of first-order sentences is an interpretation whose domain is a quotient of the set of ground terms $T(F)$ by some congruence relation and such that the interpretation of every ground term t is the congruence class of t . In such interpretations, every element of the domain can be constructed from elements in the signature F ; coming back to example 1.1, the interpretation I is not an Herbrand interpretation since a cannot be constructed from $0, s, +$ only. Inductive theorems are the formulas which can be proved using E and an axiom scheme which rules out interpretations which are not Herbrand interpretations. This leads us to the following definition:

2.1. DEFINITION. Let E be a set of first-order sentences. A first-order formula ϕ is an *inductive consequence* of E iff, for every Herbrand interpretation \mathcal{H} , $\mathcal{H} \models E$ implies $\mathcal{H} \models \phi$.

The set of all inductive consequences of E is called the *inductive theory* of E . Inductive consequences of E will also be called *inductive theorems* in what follows.

When E is a set of Horn clauses with equality, there is a (unique) smallest (minimal) Herbrand model of E , which we write \mathcal{I}_E . \mathcal{I}_E is the intersection of all Herbrand models of E (hence its minimality): for each predicate symbol P , a tuple of ground terms belongs to the interpretation of P in \mathcal{I}_E iff it is the case in all Herbrand models of E . \mathcal{I}_E is also the least fixed point of E : consider the immediate consequence operator T_E which maps a Herbrand interpretation \mathcal{H} to a Herbrand interpretation \mathcal{H}' as follows. An atom $P(t_1, \dots, t_n)$ belongs to \mathcal{H}' iff either it is in \mathcal{H} , or it is a consequence of strictly smaller (w.r.t. size) equalities in \mathcal{H}' or else there is a clause $B \Rightarrow P(u_1, \dots, u_n)$ in E and a ground substitution σ such that $\mathcal{H} \models B\sigma$ and $P(t_1, \dots, t_n) \equiv P(u_1, \dots, u_n)\sigma$. \mathcal{I}_E is the union of $T_E^n(\emptyset)$ for $n \in \mathbb{N}$. Finally, \mathcal{I}_E is also called the *initial model* of E because it is an initial object in the category of models of E .

When E is a set of Horn clauses, we do not need to consider all the Herbrand interpretations, but only the smallest one²:

2.2. LEMMA. *Let E be a set of Horn clauses and c be a positive clause. c is an inductive consequence of E iff $\mathcal{I}_E \models c$.*

When E is simply a set of (implicitly universally quantified) equations, \mathcal{I}_E is the quotient algebra $T(F)/_{=E}$.

2.3. EXAMPLE. In example 1.1, $x + 0 = x$ is not a consequence of E , but it is an inductive consequence since for every ground term t , $t + 0 =_E t$.

Inductive consequences of E should not be confused with elements in the theory of \mathcal{I}_E . For instance, $x \neq s(x)$ is *not* an inductive consequence of the set E of example 1.1 since the trivial one element model of E does not satisfy $x \neq s(x)$. However, $\mathcal{I}_E \models x \neq s(x)$. The inductive theory is contained in the theory of \mathcal{I}_E , but the converse is false. In what follows, we will prove or disprove conjectures in the theory of \mathcal{I}_E . This coincide with inductive consequences for positive conjectures only. However, we also consider the possibility of non-positive conjectures, in which case the results are also interesting, though having a different meaning. For instance, what is known in logic programming as *negation as failure* will be an instance of our proof by consistency method: if the (negative) conjecture is consistent with E (finite Failure of the derivation of an empty clause), then it is valid in \mathcal{I}_E . That is exactly what negation as failure says.

2.4. Constructors and sufficient completeness

The problem we encountered in example 1.2 is the absence of *free constructors*, i.e. the model we have in mind cannot be freely generated by (some) symbols in the alphabet. Let us make this notion more formal.

²Similarly, for arbitrary clauses, we do not need to consider all Herbrand interpretations, but only the minimal ones.

2.4. DEFINITION. A subset C of F is called a set of *constructors* (w.r.t. E) if, for every term t in $T(F)$, there is a term u in $T(C)$ such that $E \models t = u$.

2.5. EXAMPLE. $\{0, s\}$ is a set of constructors in example 1.1.

2.6. DEFINITION. E is *sufficiently complete* with respect to a subset D of F if $C = F \setminus D$ is a set of constructors w.r.t. E . Then elements of D are called *defined symbols*.

2.7. EXAMPLE. Assume that we have, in addition to example 1.1, a new function symbol p and the three additional equations:

$$\begin{cases} p(s(x)) &= x \\ s(p(x)) &= x \\ p(x) + y &= p(x + y) \end{cases}$$

Then E is sufficiently complete w.r.t. $\{+\}$.

If we know a set of constructors, then we may use a structural induction over this set of constructors in order to prove inductive consequences of E . However, this is often not sufficient: F itself is always a set of constructors. In example 1.1, if we try to prove $x + 0 = x$ by structural induction on F , it will be hard to succeed because there are a lot of necessary lemmas (such as the associativity of $+$) which are in turn hard to prove using such a structural induction.

One can argue that F is really a bad choice. We could choose a *minimal* set of constructors. However, on one hand such a minimal set is not easy to find and, on the other hand, it is not sufficient in many cases. This can be illustrated by example 1.2: $\{0, s, -\}$ is a minimal set of constructors (how do you prove it?). Trying to prove $x + 0 = x$ by structural induction on the set of constructors is not so easy, and requires some lemmas (which have to be discovered).

2.8. DEFINITION. A set C of constructors is *free* if $E \cup \{s \neq t \mid s, t \in T(C), s \neq t\}$ is consistent.

When E is a set of Horn clauses, C is a set of constructors iff for every distinct terms $s, t \in T(C)$, $E \not\models s = t$. In example 1.1, the set $C = \{0, s\}$ of constructors is *free* because, for any two distinct terms $s, t \in T(C)$, $s \neq_E t$. However, in examples 1.2 and 2.7, any set of constructors is not free.

2.5. Term Rewriting

This section and the next one are devoted to a detour through term rewriting. Though this detour is not necessary for our purpose, many examples will use this framework. We only recall very briefly the basic background. More information and developments can be found in e.g. [Bachmair 1991, Dershowitz and Jouannaud 1990]

and in [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook). The reader who is familiar with term rewriting systems can jump to section 3.

A *rewrite rule* is an oriented equation. A *rewrite system* is a set of rewrite rules. Rewriting a term t into a term u using a rule $l \rightarrow r$ with the substitution σ is defined as the replacement of equals by equals, except that only $l\sigma$ can be replaced with $r\sigma$ and not the converse. This relation is written $t \xrightarrow[l \rightarrow r]{\sigma} u$. A rewrite system \mathcal{R} defines a *rewrite relation* $\xrightarrow{\mathcal{R}}$ which is the union of all $\xrightarrow[l \rightarrow r]{\sigma}$ for all substitutions σ and rules $l \rightarrow r$ in \mathcal{R} . The corresponding *reduction relation* is the reflexive transitive closure $\xrightarrow{\mathcal{R}}^*$ of $\xrightarrow{\mathcal{R}}$.

\mathcal{R} is *confluent* if, for all terms s, t, u such that $s \xrightarrow{\mathcal{R}}^* t$ and $s \xrightarrow{\mathcal{R}}^* u$, there is a term v such that $t \xrightarrow{\mathcal{R}}^* v$ and $u \xrightarrow{\mathcal{R}}^* v$. \mathcal{R} is *terminating* iff there is no infinite sequence $t_1 \xrightarrow{\mathcal{R}} \dots \xrightarrow{\mathcal{R}} t_n \xrightarrow{\mathcal{R}} \dots$. \mathcal{R} is *convergent* when it is both terminating and confluent. When \mathcal{R} is convergent, each term t has a unique normal form $t \downarrow_{\mathcal{R}}$. For convergent rewrite systems \mathcal{R} , the equational theory is decidable: $s =_{\mathcal{R}} t$ iff $s \downarrow_{\mathcal{R}} \equiv t \downarrow_{\mathcal{R}}$. For more details see [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook).

2.6. Standard Completion

Completion processes (which are explained in more detail Chapter 9 [Dershowitz and Plaisted 2001]) consist basically in orienting equations according to a given reduction ordering and computing equational consequences. They, roughly, aim at computing from a set of equations an equivalent convergent rewrite system.

More precisely, a completion procedure takes as input a set of equations E and a reduction ordering \succeq and applies successively one of the rules given in [Dershowitz and Plaisted 2001, page 571] (Chapter 9 of this Handbook):

Deduce which adds some critical pairs

Delete, Collapse, Simplify which simplify the rules and the equations

Orient which turns equations into rules according to the ordering.

A completion procedure is fair whenever some possible deduction or orientation is not indefinitely delayed. A completion procedure may either *fail* because some unorientable and unsimplifiable equation is found (in which case the completion cannot be fair). Otherwise, the ultimate set of persisting rewrite rules R_{∞} is a convergent rewrite system (see [Dershowitz 1991, Bachmair 1991]).

Examples of completion can be found in [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook). Let us give here a simple example:

2.9. EXAMPLE. We consider the presentation of integers in example 1.2. Using a recursive path ordering extending the precedence $+ > - > s > 0$, every equation can be turned into a rewrite rule and we get the rewrite system:

$$\mathcal{R} = \begin{cases} -0 \rightarrow 0 & (1) \\ - - x \rightarrow x & (2) \\ s(-s(x)) \rightarrow -x & (3) \\ 0 + x \rightarrow x & (4) \\ s(x) + y \rightarrow s(x + y) & (5) \end{cases}$$

Overlapping (3) with itself we get

$$s(x) \xleftarrow{(2)} s(- - x) \xleftarrow{(3)} s(-s(-s(x))) \xrightarrow{(3)} - - s(x) \xrightarrow{(2)} s(x)$$

yielding a critical pair which is simplified and deleted. Overlapping (3) and (5):

$$s(-s(x) + y) \xleftarrow{(5)} s(-s(x)) + y \xrightarrow{(3)} -x + y$$

we get the critical pair $s(-s(x) + y) = -x + y$ which can be turned into a new rule:

$$s(-s(x) + y) \rightarrow -x + y \quad (6)$$

There are new critical pairs: (3) + (6) gives

$$\begin{aligned} s(x + y) &\xleftarrow{5} s(x) + y \xleftarrow{(2)} - - s(x) + y \xleftarrow{(6)} s(-s(-s(x)) + y) \xrightarrow{(3)} \\ &\xrightarrow{(3)} s(- - x + y) \xrightarrow{(2)} s(x + y) \end{aligned}$$

which is simplified and deleted. (6) + (3) gives

$$-(-s(x) + y) \xleftarrow{(3)} s(-s(-s(x) + y)) \xrightarrow{(6)} s(-(-x + y))$$

which is turned into a new rule

$$-(-s(x) + y) \rightarrow s(-(-x + y)) \quad (7)$$

(7) + (2) yields

$$-s(-(-(-x + y))) \xleftarrow{(7)} - - (-s(x) + y) \xrightarrow{(2)} -s(x) + y$$

which is turned into the new rule

$$-s(x) + y \rightarrow -s(-(-x + y)) \quad (8)$$

(6) is simplified using (8):

$$s(-s(x) + y) \xrightarrow{(8)} s(-s(-(-x + y))) \xrightarrow{(3)} - - (-x + y) \xrightarrow{(2)} -x + y$$

and then deleted. Similarly, (8) simplifies (7) which is deleted and all remaining critical pairs can be simplified and deleted. We get the convergent rewrite system:

$$\left\{ \begin{array}{lll} -0 & \rightarrow & 0 \\ - - x & \rightarrow & x \\ s(-s(x)) & \rightarrow & -x \\ 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x + y) \\ -s(x) + y & \rightarrow & -s(-(-x + y)) \end{array} \right.$$

3. General Setting of the Inductionless Induction Method

For simplicity, we assume from now on, in the rest of this chapter, that E is a finite set of Horn clauses. To some extent, it is possible to generalize the approach to arbitrary clauses, considering a *perfect model semantics*. The interested reader is referred to [Comon and Nieuwenhuis 1998].

The method roughly works as follows: given a set of clauses (with equality) E , and a set of conjectures \mathcal{C} , we add \mathcal{C} to E and try to derive an inconsistency. If this turns out to be impossible, then \mathcal{C} is an inductive consequence of E . "Inconsistency" here has to be understood w.r.t. some axiomatization \mathcal{A} of \mathcal{I}_E .

3.1. EXAMPLE. Let us consider the simplest example (example 1.1): Let \mathcal{A} be the set of (universally quantified) formulas:

$$\left\{ \begin{array}{l} s(x) \neq 0 \\ s(x) = s(y) \Rightarrow x = y \end{array} \right.$$

$\mathcal{I}_E \models \mathcal{A}$. Now, assume that we conjecture $s(x) + 0 = s(0)$. Adding this equation to E , we can derive

$$s(0) \xleftrightarrow{s(x)+0=s(0)} s(s(y)) + 0 \xleftrightarrow{E} s(s(y) + 0) \xleftrightarrow{E} s(s(y + 0)).$$

i.e. for all y , $s(0) = s(s(y + 0))$. Then using the second formula of \mathcal{A} , we get $(\forall y).s(y+0) = 0$, which contradicts the first formula of \mathcal{A} . This shows that $s(x)+0 = s(0)$ is not an inductive consequence of E .

More generally, if \mathcal{A} is a set of formulas such that $\mathcal{I}_E \models \mathcal{A}$ and if $\mathcal{A} \cup E \cup \mathcal{C}$ is inconsistent, then \mathcal{C} cannot be a set of inductive consequences of E . We also want some converse implication, allowing to derive the inductive validity of \mathcal{C} from the consistency of $\mathcal{A} \cup E \cup \mathcal{C}$. This is summarized in the following properties of \mathcal{A} :

3.2. DEFINITION. An *I-axiomatization* of \mathcal{I}_E is a recursive set \mathcal{A} of purely universal formulas such that:

1. $\mathcal{I}_E \models \mathcal{A}$
2. For every Herbrand model \mathcal{M} of E

$\mathcal{M} \models \mathcal{A}$ implies \mathcal{M} is isomorphic to \mathcal{I}_E

3.3. EXAMPLE. The set \mathcal{A} of example 3.1 is an I-axiomatization of the corresponding algebra \mathcal{I}_E .

Of course such axiomatizations are, in general, incomplete since the first-order theory of \mathcal{I}_E may be not recursive. However, they have a nice property which is the basis of the method:

3.4. LEMMA. *Let \mathcal{A} be an I-axiomatization of \mathcal{I}_E . If $E \cup \mathcal{C} \cup \mathcal{A}$ is consistent then $\mathcal{I}_E \models \mathcal{C}$. Hence, if \mathcal{C} is a set of positive clauses, it is an inductive consequence of E .*

PROOF. If $E \cup \mathcal{C} \cup \mathcal{A}$ is consistent, then it has at least one Herbrand model because it is a set of purely universal sentences. Now, by property 2 of definition 3.2, $E \cup \mathcal{C} \cup \mathcal{A}$ has at most one Herbrand model, up to isomorphism. Hence $E \cup \mathcal{C} \cup \mathcal{A}$ has exactly one Herbrand model, which is \mathcal{I}_E . \square

This is a key lemma since, roughly, it reduces (via \mathcal{A}) inductive consequences to first-order consistency.

It remains, however, to clarify several problems. First, how is it possible to check consistency (or inconsistency) of $E \cup \mathcal{C} \cup \mathcal{A}$? Then how is it possible to find I-axiomatizations?

We delay the problem of finding I-axiomatizations until the next sections (section 5 is dedicated to a number of examples for \mathcal{A}). With respect to the first issue, a first remark is that we do not need to consider inferences whose premisses are conjectures only (we may use a *linear strategy*). The reason is given by the following simple lemma:

3.5. LEMMA. *Let \mathcal{A} be an I-axiomatization. Then $\mathcal{A} \cup E \cup \mathcal{C}$ is inconsistent iff there is a clause $c \in \mathcal{C}$ and a ground substitution σ such that $c\sigma \cup \mathcal{A} \cup E$ is inconsistent.*

PROOF. Simply, if $c\sigma \cup \mathcal{A} \cup E$ is consistent, then \mathcal{I}_E is a model of these formulas. \square

At this point, we may use any refutation-complete first-order deduction system, together with a linear strategy. If \mathcal{C} is not an inductive consequence of E , then an inconsistency proof will be found. However, if \mathcal{C} is an inductive consequence of E , we should have a chance to complete the proof. In other words, we need to be able to prove the consistency. As already explained, the idea will be to saturate the set of clauses: a saturated set of clauses is consistent if and only if it does not contain the empty clause. Unfortunately, the saturation process often runs forever. Hence any restriction on the consequences which have to be computed is welcome. One example is the above linear strategy. There is also one property which is known and

not yet exploited: we know that $\mathcal{A} \cup E$ is consistent. Moreover, roughly, \mathcal{A} contains only “negative” information (stating what should be distinct in the model). We may exploit these informations. For instance, we will perform the (in)consistency proofs in two stages: first deductions on $\mathcal{C} \cup E$, yielding a possible inconsistency witness c , then the (in)consistency proof of $c \cup \mathcal{A}$.

Such additional restrictions on the consequences which have to be computed are captured in the inductive saturation strategies which we are going to introduce now.

4. Inductive completion methods

This section is devoted to the deduction engine in the framework of inductive proofs by consistency methods. Historically, the deduction engine was the Knuth-Bendix completion procedure: E was a set of equations, as well as \mathcal{C} . Therefore, all prototype implementations and published examples of this proof by consistency approach rely on the completion of term rewriting systems. We will however try to give a more general view of the deduction systems, without restricting ourselves to the equational case. As we have seen, inductive proofs can be reduced to some particular (in)consistency proofs which can be performed by any first-order theorem prover. The only problem is that, in general, such provers are designed for *refutation*, i.e. *inconsistency* proofs. We need here a first-order theorem prover which is (also) able to conclude the *consistency* of a set of formulas. Saturation based theorem provers are adequate for this purpose. The goal now is to design some dedicated strategies which can be used with a saturation-based theorem prover in the context of inductive proofs.

First, we recall the notions of redundancy and saturation in section 4.1. Next, we state a general framework of inductive completion (or inductive saturation) in section 4.2. Then we propose a set of deduction rules for inductive completion in section 4.3. What we are presenting covers the inductive completion methods described in [Bachmair 1991]; when E and \mathcal{C} are equations we find these completion techniques as particular instances of what is presented here. In section 4.4, we show that the deduction rules of section 4.3 satisfy the desired properties, provided that \mathcal{A} is a so-called “normal axiomatization” (a particular class of I -axiomatization which will be defined there). Section 4.5 is devoted to a number of examples, while section 4.6 investigates some further refinements of the strategy.

4.1. Redundancy and Saturation

The notions of redundancy and saturation are studied in detail in [Bachmair and Ganzinger 2001, page 36] (Chapter 2 of this Handbook). Roughly, a clause c is redundant in a set of clauses \mathcal{S} if all its ground instances are consequences of strictly smaller clauses in \mathcal{S} . A set of clauses is saturated (w.r.t. an inference system) if all inferences yield redundant clauses. We strengthen these notions in our framework of

proofs by consistency: we will say that a clause c is redundant in \mathcal{S} if all its ground instances are consequences of strictly smaller clauses in \mathcal{S} and any instances of inductively valid clauses. This is important since we have more redundant clauses, hence it is easier to get a saturated set.

We assume that \succeq is a reduction ordering that is total on ground terms (i.e. an ordering which is well-founded, total, and monotonic: $t \succ u$ implies $f(\dots t \dots) \succ f(\dots u \dots)$ for every function symbol f ; see [Dershowitz and Plaisted 2001], Chapter 9 of this Handbook, for more on reduction orderings). This ordering is extended into a total ordering on ground clauses, such that ground equalities are smaller than any other ground literal. . Given a ground clause C , $C^<$ is the set of ground clauses that are strictly smaller than C in this ordering.

4.1. EXAMPLE. Assume that we have the function symbols $0, s, +$ and one binary predicate symbol gt (besides equality). The ground terms can be ordered according to the lexicographic path ordering extending the precedence $+ > s > 0$: $0 < s(0) \dots < s^n(0)$, any term containing $+$ is larger than a term not containing $+$ and $s_1 + s_2 < t_1 + t_2$ iff either $s_1 < t_1$ and $s_2 < t_1 + t_2$ or $s_1 = t_1$ and $s_2 < t_2$ or else $s_1 + s_2 \leq t_2$. Then

$$s(0) + 0 = 0 < gt(0, 0) < gt(0 + (s(0) + s(0)), s(0)) < gt(s(0) + 0, 0)$$

The clause $s(0 + x) = s(x)$ is redundant in $\{0 + x = x\}$.

4.2. DEFINITION. A ground conjecture c is *redundant* in a set of conjectures \mathcal{C} if $E \cup \mathcal{A} \cup \mathcal{C}^{<c} \models c$. A non-ground conjecture is redundant if all its ground instances are redundant.

An inference is redundant in \mathcal{C} if one of its premisses or its conclusion is redundant in \mathcal{C} .

Note here that, for redundancy, we may use the axioms E and \mathcal{A} without any restriction, which makes a difference with the notions defined in [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook).

This notion can be refined, using well-known techniques in the field of saturation-based theorem proving [Bachmair and Ganzinger 1994, Nieuwenhuis and Rubio 1995]. For instance, instead of comparing the instances of the clauses, we may compare the pairs (c, σ) , which allows to capture subsumption.

4.3. EXAMPLE. Here is an ordering which, specialized to the pure equational case, gives the proof ordering of [Bachmair 1988].

We define an ordering \gg on pairs (c, σ) where c is a clause and σ is a substitution as follows: $(c, \sigma) \gg (c', \sigma')$ iff the multiset of pairs (L, σ) for atomic formulas $L \in c$ is larger than the multiset of pairs (L', σ') for atomic formulas $L' \in c'$. The multisets are compared according to the multiset extension of the following ordering:

- For every predicate symbol P distinct from the equality,

$$(P(t_1, \dots, t_n), \sigma) \gg (s = t, \sigma')$$

- for every predicate symbols P, Q distinct from the equality,
 $(P(t_1, \dots, t_n), \sigma) \gg (Q(s_1, \dots, s_m), \sigma')$ iff
 either $P(t_1, \dots, t_n)\sigma \succ Q(s_1, \dots, s_m)\sigma'$
 or else $P(t_1, \dots, t_n)\sigma = Q(s_1, \dots, s_m)\sigma'$ and $P(t_1, \dots, t_n) \succ Q(s_1, \dots, s_m)$.
- $(s = t, \sigma) \gg (u = v, \sigma')$ iff $K(s = t, \sigma) \gg K(u = v, \sigma')$ where K is defined by:

$$\begin{cases} K(s = t, \sigma) = (\{s\sigma\}, s, t\sigma) & \text{if } s \succ t \\ K(s = t, \sigma) = (\{t\sigma\}, t, s\sigma) & \text{if } t \succ s \\ K(s = t, \sigma) = (\{s\sigma, t\sigma\}, s = t, \perp) & \text{otherwise} \end{cases}$$

and triples are ordered using the lexicographic extension of the multiset extension of \succeq , the encompassment ordering, \succeq respectively. \perp is the smallest ground term.

Using such a refinement, a clause $c\sigma$ is redundant in a set of clauses which contains c because, for every literal L and every ground substitution θ , $(L\sigma, \theta) \gg (L, \sigma\theta)$.

Similarly, if an equation $s = t$ is reducible by a rule $l \rightarrow r$ (with $lsuccr$), then $s = t$ is redundant in any set of clauses containing $l = r$ and the simplified version $s' = t'$ of $s = t$. Hence we may *simplify* $s = t$, replacing it with $s' = t'$, as it is usual in term rewriting.

Finally, a set of clauses is *saturated* (with respect to a given redundancy criterion and an inference system Inf) if all inferences are redundant. In what follows, by “saturated” we will implicitly assume the inference system \mathcal{I} of [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook).

With a set of clauses E is associated a convergent rewrite system \mathcal{R}_E on ground terms defined by $s \rightarrow t \in \mathcal{R}_E$ iff $E \models s = t$ and t is minimal (w.r.t. \succeq) among the terms $\{u \mid u \neq s, E \models s = u\}$. For saturated sets of clauses, \mathcal{R}_E is the interreduced version of the definition given in [Nieuwenhuis and Rubio 2001, page 386] (Chapter 7 of this Handbook).

From results on saturated sets of clauses, we get immediately the following:

4.4. LEMMA. *If E is saturated and $E \models u = v$ for two ground terms u, v , then either $u \equiv v$ or else there is a ground instance $D \vee l = r$ of a clause in E such that $l > D, r, E \not\models D$ and $u = v$ is reducible by $l \rightarrow r$.*

4.2. Inductive saturation strategies

The first definition expresses the ability to split the (in)consistency proofs into two parts: deductions between \mathcal{C} and E on one hand and (in)consistency proofs of $\mathcal{C} \cup \mathcal{A}$ on the other hand.

4.5. DEFINITION. Let \mathcal{A} be an I -axiomatization of \mathcal{I}_E . An *inductive saturation strategy* is a mapping S which associates each pair E, C with a set of clauses \mathcal{U} such that

1. Each element of \mathcal{U} is a logical consequence of E, C
2. If $S(E, C) = \emptyset$, then C is an inductive consequence of E iff for every $c \in C$, c is consistent with \mathcal{A} .

For instance, if S computes all non-redundant consequences of E, C , then $S(E, C) = \emptyset$ means that $E \cup C$ is saturated. More generally, S may compute some of the non-redundant consequences only, provided that they are sufficient to detect inconsistencies. A typical such example is given by *linear strategies*: if we start from a set of axioms which is saturated, only overlaps of the axioms into the conjectures suffice to detect inconsistencies. This is the basis for instance of narrowing strategies (see [Slagle 1974] for an early reference or [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook). And, indeed, the idea of S is to narrow the conjectures until we find a counter-example (or no other deduction is possible).

Given an inductive saturation strategy S a *derivation sequence* is a sequence $C_0, C_1, \dots, C_n, \dots$ such that $C = C_0$ and each C_{i+1} is obtained from C_i by removing some redundant clause or by adding to C_i some $c \in S(E, C_i)$.³

A derivation sequence is *fair* (w.r.t. S) if every clause which can be persistently derived is eventually derived:

$$c \in \bigcap_{i \geq 1} \bigcup_{j \geq i} S(E, C_j) \Rightarrow \exists j, c \in C_j.$$

A reformulation of the definitions gives:

4.6. LEMMA. *Every inductive saturation strategy is refutationally complete: if C is not a set of inductive theorems, then, for every fair derivation sequence, there is an index i and a clause $c \in C_i$ such that $c, \mathcal{A} \vdash \square$.*

Now, we have to show some instances of this general scheme: we have to provide some inductive saturation strategies.

4.3. Ordered strategies

We consider a superposition calculus: we use an inference system which is a restriction of ordered paramodulation as defined in [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook). If there are non-equational atoms $P(\dots)$, we can encode them as equations $P(\dots) = \text{true}$. We prefer here to avoid the encoding for two reasons: first, examples are more naturally expressed with the predicate representation, second, additional restrictions can be put on the application of inference rules for such literals. Therefore, we are also using restricted forms of ordered resolution.

³For those who are familiar with the subject, such a definition rules out the possibility to simplify an axiom using a conjecture. This is for sake of simplicity only. We can of course derive pairs (E_i, C_i) .

In our framework, we add a strong restriction on the possible deductions: we use a linear strategy (as for so-called *narrowing*): only deductions overlapping an axiom (from E) into a conjecture are allowed. We give below the inference rules.

Conjecture superposition

$$\frac{D \vee l = r \quad C \vee L[s]}{D\sigma \vee C\sigma \vee L[r]\sigma}$$

if

- $\sigma = \text{mgu}(l, s)$,
- s is not a variable,
- $r\sigma \not\leq l\sigma$, $D\sigma \not\leq l\sigma$.
- $D \vee l = r \in E$
- $C \vee L[s] \in C$

[Equality resolution]

$$\frac{C \vee s \neq t}{C\sigma}$$

if

- $\sigma = \text{mgu}(s, t)$
- $C \vee s \neq t \in C$

Ordered binary resolution

$$\frac{D \vee P(s_1, \dots, s_n) \quad C \vee \neg P(t_1, \dots, t_n)}{D\sigma \vee C\sigma}$$

if

- $\sigma = \text{mgu}(P(s_1, \dots, s_n), P(t_1, \dots, t_n))$,
- $D \vee P(s_1, \dots, s_n) \in E$,
- $C \vee \neg P(t_1, \dots, t_n) \in C$,
- $P(s_1, \dots, s_n)\sigma \not\leq D\sigma$.

[Ordered factoring]

$$\frac{C \vee L \vee L'}{C\sigma \vee L\sigma}$$

if

- $\sigma = \text{mgu}(L, L')$
- $L\sigma$ is maximal in $(C \vee L \vee L')\sigma$
- $C \vee L \vee L' \in \mathcal{C}$

For all inference rules, the result of the inference is added to \mathcal{C} .

Remarks

- Redundancy criteria based on the same ordering can be used as well: inferences are restricted to non-redundant inferences and we may use deletion of redundant conjectures.
- In our framework, the redundancy criteria can be further refined, using in an unrestricted way formulas that are known to be valid in \mathcal{I}_E .
- Though superpositions are restricted to conjectures and axioms, we may well use both lemmas which are known to be inductive theorems and axioms in \mathcal{A} . This often yields simplifications.
- The above inference rules allow superpositions on the non-maximal sides of the equations in the conjectures. We may impose the superposition on the maximal side of negative literals without changing the results. However, the classical inductive completion methods for equations only considers superpositions on the maximal sides of equations (hence positive literals) [Bachmair 1988]. This restriction can be added to the inference system, with two provisions: we have to add an equality resolution inference rule (and an ordered factorization). Moreover, a slightly stronger property on axiomatizations is needed (see [Comon and Nieuwenhuis 1998] for more details). Then our inference system above is a generalization to the classical inductive Knuth-Bendix completion.
- Further restrictions can be imposed without altering the results. They are discussed in section 4.6

Actually, we could also imagine other inference rules and other strategies. What is important here is the idea of narrowing: we want to deduce the minimal counter-examples from false conjectures. More precisely, we want to keep the following property:

4.7. LEMMA. *Assume that E is a saturated set of Horn clauses and \mathcal{C} is a set of conjectures. Then*

1. *If $c\sigma$ is a minimal ground instance of a clause in \mathcal{C} such that $c\sigma \cup \mathcal{A} \cup E$ is inconsistent and if $c\sigma$ is reducible by \mathcal{R}_E , then there is a non-redundant inference $c, E \vdash c'$ and a ground substitution σ' such that $c'\sigma' \cup \mathcal{A} \cup E$ is inconsistent and $c'\sigma' \prec c\sigma$.*
2. *If $E \models P(s_1, \dots, s_n)\sigma$ and if $c\sigma \equiv \neg P(s_1, \dots, s_n)\sigma \vee c_1\sigma$ is a minimal ground instance of a conjecture which is irreducible by \mathcal{R}_E and such that $c\sigma \cup \mathcal{A} \cup E$ is inconsistent, then there is a non-redundant inference $c, E \vdash c'$ and a ground substitution σ' such that $c'\sigma' \cup \mathcal{A} \cup E$ is inconsistent and $c'\sigma' \prec c\sigma$.*

PROOF. (sketch) We only consider the first property: the second one is similar. σ is irreducible by minimality of $c\sigma$. Then $c\sigma$ is reducible at a non-variable position

p by a rule $l\theta \rightarrow u \in \mathcal{R}_E$. Assume moreover, without loss of generality, that $l\theta$ is irreducible by any other rule. E being saturated, by lemma 4.4, there is a clause $D\vee l = v \in E$ with $l\theta \succ v\theta, D\theta$. Then there is an inference $c, D\vee l = v \vdash c[v]_p \tau \vee D\tau$ by conjecture superposition, yielding a clause c' whose instance $c\sigma[v\theta]_p \vee D\theta$ is strictly smaller than $c\sigma$, and which is also inconsistent with $E \cup \mathcal{A}$. The inference is not redundant, otherwise $c\sigma$ itself would be redundant in \mathcal{C} , which would contradict its minimality. \square

4.4. Normal axiomatizations

4.8. DEFINITION. An I-axiomatization \mathcal{A} of \mathcal{I}_E is *normal* if for all ground terms s_1, \dots, s_n , and every predicate symbol P ,

$$\begin{cases} s_1 \downarrow_{\mathcal{R}_E} \neq s_2 \downarrow_{\mathcal{R}_E} & \text{implies } \mathcal{A} \models s_1 \downarrow_{\mathcal{R}_E} \neq s_2 \downarrow_{\mathcal{R}_E} \\ E \not\models P(s_1 \downarrow_{\mathcal{R}_E}, \dots, s_n \downarrow_{\mathcal{R}_E}) & \text{implies } \mathcal{A} \models \neg P(s_1 \downarrow_{\mathcal{R}_E}, \dots, s_n \downarrow_{\mathcal{R}_E}) \end{cases}$$

Let us give three examples of normal axiomatizations which can be obtained using the techniques described in section 5:

4.9. EXAMPLE. Back to example 1.1, the axiomatization given in example 3.1 is normal.

4.10. EXAMPLE. Consider

$$\mathcal{A} = \{s = t \Rightarrow s \equiv t \mid s, t \in T(F), N(s), N(t)\}$$

where N and \equiv are the predicates defined on $T(F)$ by the following rules:

$$\begin{cases} P(0) \\ P(s(x)) \Leftarrow P(x) \\ N(x) \Leftarrow P(x) \\ N(-x) \Leftarrow P(x) \\ f(s_1, \dots, s_n) \equiv f(t_1, \dots, t_n) \Leftrightarrow s_1 = t_1 \wedge \dots \wedge s_n = t_n & \text{for every } f \in F \\ f(s_1, \dots, s_n) \not\equiv g(t_1, \dots, t_m) & \text{for every } f \neq g \end{cases}$$

\mathcal{A} is a normal axiomatization for the example 1.2. Intuitively, the new predicate N is satisfied on ground terms in normal form w.r.t. the system of example 2.9.

4.11. EXAMPLE. $F = \{0, s\}$ with one predicate symbol $>$.

$$E = \begin{cases} s(x) > 0 \\ s(x) > s(y) \Leftarrow x > y \end{cases}$$

is supposed to define the strict ordering on natural numbers.

$$\mathcal{A} = \begin{cases} \neg(0 > x) \\ \neg(s(x) > s(y)) \Leftarrow \neg(x > y) \end{cases}$$

is a normal axiomatization of \mathcal{I}_E .

If we let \mathcal{S}_N be defined by the rules of the previous section, then the main result is the following:

4.12. THEOREM. *If E is a finite saturated set of Horn clauses and if \mathcal{A} is a normal axiomatization, then \mathcal{S}_N is an inductive saturation strategy.*

PROOF. (sketch). Assuming that $\mathcal{A} \cup E \cup \mathcal{C}$ is inconsistent and that $\mathcal{S}_N(E, \mathcal{C}) = \emptyset$, we will prove that there is a clause $c \in \mathcal{C}$ and a substitution σ such that $c\sigma \cup \mathcal{A}$ is inconsistent. By lemma 3.5, we know that there is at least one c and one σ such that $c\sigma \cup E \cup \mathcal{A}$ is inconsistent. We choose them in such a way that $c\sigma$ is minimal among ground instances of clauses in \mathcal{C} which are inconsistent with $E \cup \mathcal{A}$. Then, for every literal L of $c\sigma$, $\mathcal{I}_E \not\models L$. Since $\mathcal{S}_N(E, \mathcal{C})$ is redundant, all inferences $c, E \vdash c'$ are redundant. Hence, by lemma 4.7, $c\sigma$ is irreducible by \mathcal{R}_E : all terms occurring in L are irreducible by \mathcal{R}_E . We investigate all possibilities for L :

- if L is a literal $P(s_1, \dots, s_n)$, $E \not\models L$ by hypothesis, hence $\mathcal{A} \models \neg L$ by definition and $\mathcal{A} \cup L$ is inconsistent
- if L is a literal $\neg P(s_1, \dots, s_n)$, by lemma 4.7, and since $\mathcal{S}_N(E, \mathcal{C}) = \emptyset$, $E \not\models P(s_1, \dots, s_n)$, hence $\mathcal{A} \models L$; this case cannot occur.
- if L is an equation $s = t$, by definition of \mathcal{A} , $s = t \cup \mathcal{A}$ is inconsistent
- if L is a negated equation $s \neq t$, then $\mathcal{I}_E \models s = t$ and, since they are both in normal form, $s \equiv t$. Hence L alone is inconsistent.

□

4.5. Examples

For historical reasons, in most examples of the proof by consistency method, the set of axioms is a set of equations which can be turned into a ground convergent rewrite system. A ground convergent rewrite system is actually a saturated set of clauses. Instead of redundancy, one uses so-called simplification and deletion rules, which are particular instances of redundancy criteria. The deduction rules in this case consist in critical pairs computation, which again is a particular case of superposition.

4.13. EXAMPLE. Let us go on with the toy example 1.1. We choose for \succeq for instance the lexicographic path ordering extending the precedence $+ > s > 0$. E is turned into a convergent set of rules:

$$\begin{cases} (1) & 0 + x & \rightarrow & x \\ (2) & s(x) + y & \rightarrow & s(x + y) \end{cases}$$

hence saturated. We are going to consider 4 different conjectures which illustrate different facets of the deduction methods.

Consider the conjecture

$$(c_1) \quad x + y = y + x$$

There are four possible inferences, overlapping (c_1) with (1) or (2). We get first

$$\begin{array}{lll} (c_{1,1}) & x + 0 & \rightarrow x \\ (c_{1,2}) & y + s(x) & \rightarrow s(x + y) \end{array}$$

the arrow indicating an ordering between the two members of the equations. The two other inferences yield renamings of the two above equations, which are therefore redundant. Overlapping E on the two new conjectures, we get 4 new conjectures:

$$\begin{array}{lll} (c_{1,3}) & 0 & = 0 \\ (c_{1,4}) & s(x + 0) & \rightarrow s(x) \\ (c_{1,5}) & s(x + 0) & \rightarrow s(x) \\ (c_{1,6}) & s(y + s(x)) & = s(x + s(y)) \end{array}$$

The four new clauses are all redundant. This is obvious for $(c_{1,3})$. $(c_{1,4})$ and $(c_{1,5})$ are identical. Hence we only have to perform two redundancy proofs. Consider any ground substitution σ . $(c_{1,4}, \sigma) \gg (c_{1,1}, \sigma)$ and $c_{1,1}\sigma \models c_{1,4}\sigma$, which proves the redundancy of $(c_{1,4})$. Now $(c_{1,6}, \sigma) \gg (c_{1,2}, \sigma), (c_{1,2}, \sigma'), (c_{1,1}, \sigma'')$ where $\sigma' = \{x \mapsto y\sigma; y \mapsto x\sigma\}$ and $\sigma'' = \sigma$. Moreover,

$$s(y + s(x))\sigma \xrightarrow[c_{1,2}]{\sigma} s(s(x + y))\sigma \xrightarrow[c_1]{\sigma'} s(s(y + x)) \xrightarrow[c_{1,2}]{\sigma'} s(x + s(y))\sigma$$

which shows that $c_{1,2}\sigma, c_{1,2}\sigma', c_{1,1}\sigma'' \models c_{1,6}\sigma$, hence $c_{1,6}$ is redundant. This redundancy proof can also be seen as simplifying the two members of $(c_{1,6})$ using $c_{1,2}$, then using c_1 on a subterm.

Hence $\{c_1, c_{1,1}, c_{1,2}\} \cup E$ is saturated. We only have to check that each of these clauses is consistent with \mathcal{A} (see example 3.1), which is the case.

Consider now the conjecture

$$(c_2) \quad x + y = 0 \Rightarrow x = 0$$

As before, overlapping E into c_2 yields two clauses

$$\begin{array}{lll} (c_{2,1}) & y = 0 & \Rightarrow 0 = 0 \\ (c_{2,2}) & s(x + y) = 0 & \Rightarrow s(x) = 0 \end{array}$$

The clause $c_{2,1}$ is a tautology, hence redundant. The clause $c_{2,2}$ is not redundant, according to our definition. Then the saturation process will go on forever, generating the clauses:

$$(c_{2,2n}) \quad s^n(x + y) = 0 \Rightarrow s^n(x) = 0$$

This could have been avoided, noticing that $\mathcal{A} \models (c_{2,2})$. Such redundancy criteria could be incorporated in the method. However, the example shows some limitations, as it does not succeed on this simple example.

Consider now the conjecture

$$(c_3) \quad x = 0 \Rightarrow x + y = 0$$

Overlapping E into (c_3) we get the two clauses:

$$\begin{array}{ll} (c_{3,1}) & 0 = 0 \Rightarrow y = 0 \\ (c_{3,2}) & s(x) = 0 \Rightarrow s(x + y) = 0 \end{array}$$

$(c_{3,2})$ is redundant (using $c_{3,1}$), as well as c_3 ; $E \cup c_{3,1}$ is saturated. But, of course, $c_{3,1} \cup \mathcal{A}$ is inconsistent, as can be seen after one resolution step. Hence c_3 is not an inductive theorem.

Consider now the conjecture

$$(c_4) \quad x \neq s(x)$$

There is not any possible inference rule here: $E \cup (c_4)$ is saturated. Since c_4 is consistent with \mathcal{A} , c_4 is valid in \mathcal{I}_E . (It is not an inductive theorem however, as this is a negative conjecture).

4.6. Further refinements

4.6.1. Selection strategies

It is possible to further refine the strategy which has been presented in section 4.3. For instance, we may use a fair selection function among maximal atoms. Let us illustrate this with an example.

4.14. EXAMPLE. We continue the example 4.11. A classical conjecture is the transitivity of the ordering:

$$(c_1) \quad x > y \wedge y > z \Rightarrow x > z$$

The selection strategy marks the maximal negative literals when none of them is marked. Then one of the marked literals is selected. Marking is inherited by non-selected literals along inferences. This refinement is compatible with the previous strategy: lemma 4.7 still holds with this additional restriction.

(c_1) contains two maximal negative literals which are marked:

$$(c_1) \quad \underline{x > y} \wedge \underline{y > z} \Rightarrow x > z$$

We choose one (and only one) of them for the inference. We get (by resolution) two new conjectures:

$$\begin{array}{lcl} (c_{1,1}) & 0 > z & \Rightarrow s(x) > z \\ (c_{1,2}) & x > y \wedge \underline{s(y) > z} & \Rightarrow s(x) > z \end{array}$$

There is not any possible deduction with $c_{1,1}$. E can be overlapped on $c_{1,2}$ yielding a third conjecture:

$$(c_{1,3}) \quad x > y \wedge y > z \Rightarrow s(x) > s(z)$$

This last clause is redundant, using (c_1) and the second axiom of E . Then $E \cup \{c_1, c_{1,1}, c_{1,2}\}$ is saturated. It remains to check consistency with \mathcal{A} , which is even simpler.

Note that, on this example, without a selection strategy, considering always the overlap with the leftmost literal, we would enter an infinite loop generating the clauses

$$x > y \wedge s^n(y) > z \Rightarrow s^n(x) > z$$

4.6.2. Complete sets of positions

It is also possible to restrict the overlaps that have to be considered to *inductively complete positions* ([Fribourg 1989]).

4.15. DEFINITION. A position p in a term t is *inductively complete* if, for every ground substitution σ which is irreducible w.r.t. \mathcal{R}_E , the subterm of $t\sigma$ at position p is reducible by \mathcal{R}_E .

4.16. EXAMPLE. Let us consider yet another conjecture following example 4.13.

$$(c_5) \quad (x + y) + z = x + (y + z)$$

This conjecture can be proved without any refinement of the inductive completion strategy if the status of $+$ is chosen from left to right. Suppose however that we did a mistake and that we chose a status right-left. Then there are a priori two possible overlaps on $x + (y + z)$. However, we have seen that $+$ is a defined symbol, hence every position of a $+$ is inductively complete. Then we may consider either of the two positions: the topmost one or position 2 for the superposition. But both positions do not have to be considered. This yields in few steps a saturated set of formulas, proving that c_5 is an inductive theorem.

Assuming the following axioms defining $*$:

$$\begin{cases} 0 * x &= 0 \\ s(x) * y &= y + x * y \end{cases}$$

the saturation strategy succeeds in proving the following conjectures:

$$\begin{cases} x * (y + z) &= x * y + x * z \\ x * y &= y * x \\ x * (y * z) &= (x * y) * z \end{cases}$$

as well as a few other extensions, as described in [Bachmair 1991].

More generally, it is possible to restrict the overlaps to *inductively complete sets of positions*, as shown in [Küchlin 1989]. A set $\{p_1, \dots, p_n\}$ is an inductively complete set of positions in t if for every irreducible substitution σ one of the subterms of $t\sigma$ at positions p_1, \dots, p_n is reducible.

4.6.3. Non-saturated sets of axioms

Another generalization consists in using built-in theories such as associativity and commutativity. The deduction system can be modified, taking into account such theories, along the lines described in [Nieuwenhuis and Rubio 2001, page 421] (Chapter 7 of this Handbook). The main difficulty is to design a normal axiomatization in this context.

Finally, the main remaining restriction is the hypothesis on saturatedness of E . This hypothesis can be weakened too [Gramlich 1990b, Comon and Nieuwenhuis 1998].

5. Examples of Axiomatizations \mathcal{A} from the Literature

Up to now, we have set up a very general framework for the proof by consistency (or *inductionless induction*) method. However, historically, the method was not explained in this way. The method was actually designed by implicitly (I mean: this was hidden in the inconsistency detection rules) assuming some specific I-axiomatization. In this section, some examples of proof by consistency techniques which can be found in the literature are given. For each of these examples, we will make explicit the (normal) I-axiomatization \mathcal{A} .

Each example is characterized by

1. the hypotheses on E
2. the (normal) I-axiomatization
3. the deduction method (not always an inductive saturation strategy)
4. the decision technique for checking the consistency of $\{s = t\} \cup \mathcal{A}$

5.1. Musser's approach [Musser 1980]

At about the same time (1980), D. Musser [Musser 1980] and J. Goguen [Goguen 1980] proposed the following method.

F is assumed to contain a particular function symbol eq and two constants *true* and *false*. (Actually, we would need a particular *sort* for the Booleans, but we skip this irrelevant aspect here). E is a finite set of equations and it is assumed that, for every ground terms s, t

$$\begin{aligned} s =_E t &\Leftrightarrow eq(s, t) =_E \text{true} \\ s \neq_E t &\Leftrightarrow eq(s, t) =_E \text{false} \end{aligned}$$

1. E is a finite convergent rewrite system such that $s =_E t$ iff $eq(s, t) =_E \text{true}$ and $s \neq_E t$ iff $eq(s, t) =_E \text{false}$.
2. $\mathcal{A} = \{\text{true} \neq \text{false}\}$
3. The deduction engine is given by the Knuth-Bendix completion (assuming it does not fail) and any reduction ordering in which true and false are smaller than the other function symbols.
4. Inconsistency detection is given by the only rule:

$$\text{true} = \text{false}, \text{true} \neq \text{false} \vdash \square$$

Figure 1: Musser's Approach

That is, eq is “completely defined”.

Then, running a completion procedure, a contradiction (and therefore a disproof) is detected when an equation $\text{true} = \text{false}$ is generated. On the other hand, $s = t$ is an inductive consequence of E if $E \cup \{s = t\}$ can be completed into a convergent rewrite system without generating the inconsistent equation $\text{true} = \text{false}$. This approach can be seen as imposing an equational I -axiomatization, which is part of E itself. Then \mathcal{A} simply reduces to $\{\text{true} \neq \text{false}\}$,

Actually, these results are straightforward. Indeed, choosing \mathcal{A} , the equational deduction method and the inconsistency detection as in figure 1, they are consequences of the following proposition:

5.1. PROPOSITION. *Under the above assumptions, \mathcal{A} is an I -axiomatization.*

PROOF. By hypothesis, $\text{true} \neq_E \text{false}$, therefore $\mathcal{I}_E \models \mathcal{A}$. Now, let \mathcal{M} be an Herbrand model of E and s, t be two ground terms. There are four cases:

1. $\mathcal{M} \models s = t$ and $s =_E t$
2. $\mathcal{M} \not\models s = t$ and $s \neq_E t$
3. $\mathcal{M} \not\models s = t$ and $s =_E t$. In this case $\mathcal{M} \not\models E$.
4. $\mathcal{M} \models s = t$ and $s \neq_E t$. In this case, $eq(s, t) =_E \text{false}$. Therefore, either $\mathcal{M} \not\models E$ or $\mathcal{M} \models eq(s, t) = \text{false}$. In the latter case, $\mathcal{M} \models \text{true} = eq(s, s) = eq(s, t) = \text{false}$ and therefore $\mathcal{M} \not\models \mathcal{A}$.

As a conclusion, if $\mathcal{M} \models E \cup \mathcal{A}$, then, for all ground terms s , and t . $s =_E t$ iff $\mathcal{M} \models s = t$. \square

Note that the axiomatization is not normal since, for two ground terms s, t such that $s \neq_E t$, we don't necessarily have $\text{true} \neq \text{false} \models s \downarrow_{\mathcal{R}_E} \neq t \downarrow_{\mathcal{R}_E}$. This can be recovered as follows: let \mathcal{D} be the equations defining eq . Remove \mathcal{D} from E and

let $\mathcal{A} = \mathcal{D} \cup \{s \neq t \Leftrightarrow eq(s, t) = false\}$. Then \mathcal{A} is a normal I -axiomatization. This is more in the spirit of Musser's technique as, roughly, \mathcal{D} is an equational I -axiomatization.

5.2. EXAMPLE. Consider again example 1.1. The following set of rules completely defines eq :

$$\left\{ \begin{array}{ll} eq(0, s(x)) & \rightarrow false \\ eq(s(x), 0) & \rightarrow false \\ eq(s(x), s(y)) & \rightarrow eq(x, y) \\ eq(x, x) & \rightarrow true \end{array} \right.$$

Then the method will disprove e.g. $s(x) + 0 = x$ as follows: add $s(x) + 0 = x$ to the set of rules. This gives the rewrite system:

$$\left\{ \begin{array}{ll} 0 + x & \rightarrow x \quad (1) \\ s(x) + y & \rightarrow s(x + y) \quad (2) \\ eq(0, s(x)) & \rightarrow false \quad (3) \\ eq(s(x), 0) & \rightarrow false \quad (4) \\ eq(s(x), s(y)) & \rightarrow eq(x, y) \quad (5) \\ eq(x, x) & \rightarrow true \quad (6) \\ s(x) + 0 & \rightarrow x \quad (7) \end{array} \right.$$

Simplifying (7) with (2), (7) is replaced with

$$s(x + 0) \rightarrow x \quad (8)$$

Overlapping (1) and (8) we get a new rule

$$s(0) \rightarrow 0 \quad (9)$$

Overlapping (4) and (9) we get

$$eq(0, 0) \rightarrow false \quad (10)$$

Overlapping (6) and (9) we get finally $true = false$: the system is inconsistent and hence the conjecture is false.

Now, consider the conjecture $x + 0 = x$. Completion of the above system, replacing the conjecture $s(x) + 0 \rightarrow x$ with the new conjecture $x + 0 \rightarrow x$ yields the convergent system:

$$\left\{ \begin{array}{ll} 0 + x & \rightarrow x \\ s(x) + y & \rightarrow s(x + y) \\ eq(0, s(x)) & \rightarrow false \\ eq(s(x), 0) & \rightarrow false \\ eq(s(x), s(y)) & \rightarrow eq(x, y) \\ eq(x, x) & \rightarrow true \\ x + 0 & \rightarrow x \end{array} \right.$$

1. C is a set of free constructors. E is a finite convergent rewrite system.
- 2.

$$\mathcal{A} = \begin{aligned} & \{f(x_1, \dots, x_n) = f(y_1, \dots, y_n) \Rightarrow x_1 = y_1 \wedge \dots \wedge x_n = y_n \mid f \in C\} \\ & \cup \{f(x_1, \dots, x_n) \neq g(y_1, \dots, y_m) \mid f, g \in C, f \neq g\} \end{aligned}$$

3. equational deduction is given by the Knuth-Bendix completion procedure in which constructor (ground) terms are assumed to be smaller than any non-constructor term.
4. Inconsistency detection consists of using \mathcal{A} to decompose equations headed with constructor symbols.

Figure 2: Huet and Hullot's approach

true and *false* are still distinct, hence the conjecture is consistent with \mathcal{A} : it is an inductive theorem.

Now, of course, this approach is much too restrictive in many respects. For example, the requirement that an equality predicate is completely specified is much too strong:

5.3. EXAMPLE. Assume that F consists of 0 (constant), s, p (unary) and E contains the two equations

$$\begin{cases} s(p(x)) & = & x \\ p(s(x)) & = & x \end{cases}$$

It is not possible to design a finite set of equations E' which completely defines eq (in the above sense). (This is left as an exercise).

Another weakness is the use of standard completion which may fail: this deduction system is not an inductive saturation strategy. It is not refutationally complete.

5.2. Huet and Hullot's method [Huet and Hullot 1982]

In their 1982 paper Huet and Hullot proposed another variant of the method. They assume that the set of function symbols F contains a subset C of free constructors. The main components of the method are summarized in figure 2. Such a formulation was proposed by L. Fribourg as early as 1984 in [Fribourg 1984].

For example, compare the set \mathcal{A} above with the set given in example 3.1.

5.4. PROPOSITION. *Under the above hypotheses, \mathcal{A} is a normal I-axiomatization.*

PROOF. The recursiveness of \mathcal{A} is obvious. Assume that $x_1, \dots, x_n, y_1, \dots, y_m \in T(F)$ and $f(x_1, \dots, x_n) =_E f(y_1, \dots, y_m)$ for some $f \in C$. We may actually assume that $x_1, \dots, x_n, y_1, \dots, y_m \in T(C)$ because, by hypothesis, C is a set of constructors, hence each term in $T(F)$ can be proved to be equal to a term in $T(C)$. Then, $f(x_1, \dots, x_n)$ and $f(y_1, \dots, y_m)$ are in $T(C)$. Since the constructors are free, they are equal iff they are identical. This shows that the first set of axioms in \mathcal{A} is satisfied in $T(F)/=_E$. The verification is similar for the second set of axioms: $\mathcal{I}_E \models \mathcal{A}$.

Assume now that \mathcal{M} is an Herbrand model of $\mathcal{A} \cup E$. Let $s, t \in T(F)$. Since $\mathcal{M} \models E$, and by definition of a set of constructors, there are two terms $s' \equiv f(s_1, \dots, s_n)$ and $t' \equiv g(t_1, \dots, t_m)$ such that $s', t' \in T(C)$ and $\mathcal{M} \models s = s' \wedge t = t'$. And, since $\mathcal{M} \models \mathcal{A}$, s' and t' must be identical as soon as s and t are equal. This means that $\mathcal{M} \models s = t$ implies $E \vdash s = t$: \mathcal{M} is isomorphic to \mathcal{I}_E .

\mathcal{A} is normal since constructor terms are irreducible. \square

5.5. EXAMPLE. Let us consider again the toy example 1.1. $x+0 = x$ is an inductive theorem since, adding the rule $x+0 \rightarrow x$ to the two original rules $0+x \rightarrow x$ and $s(x)+y \rightarrow s(x+y)$ we get a convergent term rewriting system which is consistent with \mathcal{A} .

The next example was given by Huet and Hullot [Huet and Hullot 1982]. They were able to show for instance that $rev(rev(x)) = x$ automatically, without introducing new lemmas, which was required in explicit induction techniques (typically one needs the associativity of $@$).

5.6. EXAMPLE. The set of function symbols consists of

$$0, s, +, nil, cons, @, rev, length.$$

We have to assume a *sort structure* (which was not introduced so far for the sake of simplicity): $0, s, +$ take only integers as arguments and return integers. $nil, @, rev$ take only lists as arguments and return lists. $cons$ takes one integer and a list and returns a list. L takes a list as argument and returns an integer.

$$\left\{ \begin{array}{lll} (1) & 0 + x & \rightarrow x \\ (2) & s(x) + y & \rightarrow s(x + y) \\ (3) & nil @ x & \rightarrow x \\ (4) & cons(n, x) @ y & \rightarrow cons(n, x @ y) \\ (5) & length(nil) & \rightarrow 0 \\ (6) & length(cons(n, x)) & \rightarrow s(length(x)) \\ (7) & rev(nil) & \rightarrow nil \\ (8) & rev(cons(n, x)) & \rightarrow rev(x) @ cons(n, nil) \end{array} \right.$$

The specification is sufficiently complete w.r.t. $\{+, @, rev, length\}$

Using the method described above, we can automatically prove the conjectures $length(x @ y) = length(x) + length(y)$, $length(rev(x)) = length(x)$, $rev(rev(x)) = x$

Actually the two approaches (Musser's approach and Huet and Hullot's approach) are equivalent:

5.7. LEMMA. *Huet and Hullot's conditions are equivalent to Musser's conditions. In other words, any specification containing a set of free constructors can be completed to a conservative extension in which the equality predicate is completely defined.*

Remarks

1. Though not necessary, it is also possible to use \mathcal{A} in the completion process. This remark applies for other approaches as well. Actually, in their paper, Huet and Hullot add the following rule to the completion rules:

$$E \cup \{c(t_1, \dots, t_n) = c(u_1, \dots, u_n)\}; R \Rightarrow E \cup \{t_1 = u_1, \dots, t_n = u_n\}; R$$

if $c \in C$. Which corresponds to the use of an axiom of \mathcal{A} .

2. In view of the examples 1.2 and 5.3, this approach is still too restrictive. We will see in the next section how it can be extended.
3. Another weakness is again the use of standard completion which may fail.

5.3. Jouannaud and Kounalis approach [Jouannaud and Kounalis 1989]

We show here a slight extension of the Jouannaud and Kounalis and of Dershowitz [1989] approach. Now, we do not assume any set of free constructors nor any equality predicate. Instead, we assume that the axioms of the specifications are oriented into a finite (ground) convergent rewrite system \mathcal{R}_0 . Examples 3.1 and 2.7 are easy to complete in order to satisfy this condition.

\mathcal{A} contains all disequalities $s \neq t$ for terms s, t such that $s \succ t$ and s is not reducible. \mathcal{A} is, a priori, infinite. Using it in an effective way relies on the key notion of *ground reducibility*.

5.8. DEFINITION. A term t is *ground reducible* (w.r.t. \mathcal{R}_0) if all its ground instances are reducible.

Let Red be the ground reducibility predicate, which holds true on a term s when all ground instances of s are reducible. Red can be defined using a finite set of Horn clauses. As shown in section 5.5 below, $\neg Red$ can also be defined by a finite set of Horn clauses $\mathcal{S}_{\neg Red}$.

It is not very difficult to see that an equation $s = t$ such that $s \succ t$ is consistent with \mathcal{A} if and only if $Red(s)$ is consistent with $\mathcal{S}_{\neg Red}$. Such a consistency test can be performed by any first-order theorem prover. It turns out however that Red is recursive (in other words, the consistency of $\{Red(s)\} \cup \mathcal{S}_{\neg Red}$ is decidable), but this is quite difficult to prove and section 6 is devoted to this question and the feed-back on some (other) possible axiomatizations. For the moment, we see Red as a black box and assume that \mathcal{A} is a recursive set of purely universal formulas indeed.

The main components of the methods are described in figure 3.

1. E is given by a finite ground convergent rewrite system \mathcal{R}_0 .
- 2.

$$\mathcal{A} = \{s \neq t \mid s, t \in T(F), s \succ t, \neg \text{Red}(s)\}$$

3. Equational deduction is given by the Knuth-Bendix completion procedure.
4. Inconsistency detection is given by the ground reducibility test.

Figure 3: Jouannaud and Kounalis approach [Jouannaud and Kounalis 1986]

5.9. PROPOSITION. \mathcal{A} is a normal I -axiomatization.

PROOF. Assume that for all ground substitution σ , $s\sigma =_E t\sigma$. Then, because \mathcal{R}_0 is convergent, $s\sigma \downarrow_{\mathcal{R}_0} \equiv t\sigma \downarrow_{\mathcal{R}_0}$ which shows that either $s\sigma$ is reducible or $t\sigma$ is reducible or the two terms are syntactically equal. If, $s\sigma \succ t\sigma$, we are necessary in the first case: $s\sigma$ is reducible. Hence $\mathcal{I}_E \models \mathcal{A}$.

Let E be the set \mathcal{R}_0 where the rules are considered as (unoriented) equations. Assume $T(F)/\sim$ satisfies $\mathcal{A} \cup E$. Let s, t be two ground terms such that $s \sim t$. We prove by induction w.r.t. the reduction ordering $\rightarrow_{\mathcal{R}_0}^*$ on s, t that $s =_E t$. If s, t are irreducible w.r.t. $\rightarrow_{\mathcal{R}_0}$, then, by \mathcal{A} , s and t must be identical, hence $s =_E t$. Now, assume that s or t is reducible. For example $s \rightarrow_{\mathcal{R}_0} s_1$. Then $s_1 \sim t$ since $T(F)/\sim$ satisfies E . By induction hypothesis, $s_1 =_E t$, hence $s =_E t$.

By definition of \mathcal{A} , it is normal: two distinct ground terms are comparable w.r.t. \succ and, if they were equal, this would imply the reducibility of the largest one. \square

5.10. EXAMPLE. The following example cannot be proved directly by explicit induction without introducing new lemmas. It can however be automatically proved by the inductionless induction method. As it can be easily seen, it does not fit in Huet and Hullot's approach.

We consider the following axioms

$$\begin{cases} (1) & 0 + x & = & x \\ (2) & s(x) + y & = & s(x + y) \\ (3) & s(s(0)) & = & 0 \end{cases}$$

Let us show for instance that $x + x = 0$ is an inductive consequence of the axioms. First, we have to fulfill the requirements, hence to complete the system (1),(2),(3) into a convergent rewrite system. We get

$$(4) \quad s(s(x)) \rightarrow x$$

by overlapping (2) and (3), which subsumes (3). Then the system is convergent. Next, we add the conjecture

$$(5) \quad x + x \rightarrow 0$$

Overlapping (5) and (2) we get

$$(6) \quad s(x + s(x)) \rightarrow 0$$

Overlapping (4) and (6), we get

$$(7) \quad x + s(x) \rightarrow s(0)$$

which simplifies (6). Then (1),(2),(4),(5),(7) is a convergent system. Moreover, $x + x$ and $x + s(x)$ are ground reducible (i.e. they satisfy the predicate *Red* for all values of x ; this result should be assumed until we see how it is proved automatically in section 6), hence this system is compatible with \mathcal{A} : this shows that $x + x = 0$ is indeed an inductive theorem.

Now, consider the conjecture $x + s(x) = 0$. Overlapping $x + s(x) \rightarrow 0$ and (2) we get, after simplification, $s(0) = 0$ which yields a contradiction since $s(0)$ is not ground reducible: $x + s(x) = 0$ is not an inductive theorem.

The following example is a bit more complex. It describes a group generated by a, b where a has order 3 and b order 2. It is not possible to define a set of free constructors, since $*$ is not free and it occurs in all terms of some equivalence classes.

5.11. EXAMPLE. Function symbols are constants a, b, e , the binary symbol $*$ and the unary symbol $^{-1}$.

$$\left\{ \begin{array}{ll} e^{-1} \rightarrow e & a^{-1} \rightarrow a * a \\ e * x \rightarrow x & x * e \rightarrow x \\ (x^{-1})^{-1} \rightarrow x & x^{-1} * x \rightarrow e \\ x * x^{-1} \rightarrow e & (x * y)^{-1} \rightarrow y^{-1} * x^{-1} \\ (x * y) * z \rightarrow x * (y * z) & a * (a * a) \rightarrow e \\ x^{-1} * (x * y) \rightarrow y & x * (x^{-1} * y) \rightarrow y \\ a * (a * (a * x)) \rightarrow x & b * b \rightarrow e \end{array} \right.$$

The system is convergent and allows to disprove automatically, for instance $x * (x * (x * (x * (x * x)))) = e$ (i.e. every element has order 6) and gives a counter-example. Adding the axiom $a * (a * b) = b * a$, we may however prove that every element has order 6.

In this section, we gave only positive examples. There are however several weaknesses:

1. E is assumed to be given by a finite ground rewrite system \mathcal{R}_0 .
- 2.

$$\mathcal{A} = \{s \neq t \mid s, t \in T(F), \neg \text{Red}(s), s \succ t\}$$

3. Equational deduction is given by a linear restriction of the Knuth-Bendix completion procedure.
4. Inconsistency detection is given by the (co)-ground reducibility test.

Figure 4: Bachmair's approach

1. First we must meet the condition: the axioms should be turned into a convergent rewrite system. This is a strong restriction and many examples do not fall into this category.
2. Adding the conjecture, the completion may not terminate. This is unavoidable since, otherwise, the set of inductive theorems would be recursive.
3. We cannot prove unorientable conjectures such as commutativity since the standard completion fails in this case.

All these drawbacks actually come from the deduction system (Knuth-Bendix completion).

5.4. Bachmair's approach [Bachmair 1988]

The first refutationally complete inductive completion method was proposed by Fribourg in [Fribourg 1984]. Bachmair's method is the first one which does not assume a constructor discipline and which is also refutationally complete, hence is an inductive saturation strategy, following our terminology.

Assumptions on the axioms are the same as in the previous section. The axiomatization is also the same. The differences lie in the proof strategies. As in section 4, only overlaps of axioms on conjectures are necessary and conjectures need not be oriented. This is referred to as a *linear strategy* and was first proposed in [Fribourg 1986]. The main ingredients of the method are given in figure 4.

Though the axiomatization is the same as in the previous section, it is not used exactly in the same way, however; for each generated equation $s = t$, either $s \succ t$ (resp. $t \succ s$), then the consistency of $\{s = t\} \cup \mathcal{A}$ reduces to the consistency of $\{\text{Red}(s)\} \cup \mathcal{S}_{\neg \text{Red}}$, as before. Or else s and t are incomparable in the ordering. In this case, the consistency of $s = t$ with \mathcal{A} reduces to the consistency of the following finite set of clauses:

$$\begin{cases} \text{Red}(s) \vee \text{Red}(t) \vee s \equiv t \\ \mathcal{S}_{\neg \text{Red}} \\ \mathcal{S}_{\equiv} \end{cases}$$

where $S_{\neg\equiv}$ is a finite set of Horn clauses defining the negation of the syntactic equality on ground terms.

Now, in the equational deduction, conjectures need *not* to be oriented. We get something which looks like the inductive completion procedure of section 4.2 in the pure equational case: if an equation is orientable, then only superposition on the maximal side have to be considered. If the equation is not orientable, then superpositions on both sides have to be computed. This is the main idea of *unfailing completion* which is described in [Nieuwenhuis and Rubio 2001, page 375] (Chapter 7 of this Handbook). Examples of this method are for instance the commutativity of addition in example 4.13 which cannot be handled by Jouannaud and Kounalis method.

Though Bachmair's method answers one of the limitations of the Jouannaud and Kounalis one (now the completion cannot fail any more), the two other limitations remain: the original set of axioms has to be ground convergent and the saturation may not terminate.

5.5. Further examples of normal axiomatizations

Other instances of the proof by consistency approach and normal axiomatizations computations are described in [Comon and Nieuwenhuis 1998]. Some of them do not assume the saturatedness of E . For instance, we may add to E any sufficiently complete definition E' of a new function symbol; if \mathcal{A} is a normal axiomatization w.r.t. E , it remains a normal axiomatization w.r.t. E' , provided that the ground terms containing the new function symbol are larger than the terms which do not contain it. The saturatedness of E is not essential for the normal axiomatization. It is important for the inductive completion strategy, as we have seen.

5.12. EXAMPLE. Consider example 1.1 and assume that we enrich the specification with the definition of the new function symbol *gcd*:

$$E' = \begin{cases} \gcd(0, x) & = x \\ \gcd(x, 0) & = x \\ \gcd(x + y, x) & = \gcd(y, x) \\ \gcd(x, x + y) & = \gcd(x, y) \end{cases}$$

Then

$$\mathcal{A} = \begin{cases} 0 \neq s(x) \\ s(x) = s(y) \Rightarrow x = y \end{cases}$$

remains a normal axiomatization of $E \cup E'$.

In the case of Horn clauses without equality, if, in each clause, the body of the clause does not contain variables which do not occur in the head (this will be called hereafter the *variable occurrence condition*), then it is possible to compute a normal

axiomatization, using quantifier elimination for the first-order theory of finite trees [Comon 1991]. Let us show it on a very simple example:

5.13. EXAMPLE. Let E be

$$E = \begin{cases} E(0). \\ E(s(s(x))) \Leftarrow E(x). \end{cases}$$

The program is rewritten as

$$E(x) \Leftarrow x = 0 \vee (\exists y. x = s(s(y)) \wedge E(y))$$

In the least fixed point of the definition, the converse implication also holds, which gives by negating the two members:

$$\neg E(x) \Leftarrow x \neq 0 \wedge (\forall y. x \neq s(s(y)) \vee \neg E(y))$$

and, by so-called disunification:

$$\neg E(x) \Leftarrow (x = s(0) \vee (\exists y. x = s(s(y)) \wedge \neg E(y)))$$

which is turned into the normal axiomatization \mathcal{A} :

$$\mathcal{A} = \begin{cases} \neg E(s(0)). \\ \neg E(s(s(x))) \Leftarrow \neg E(x) \end{cases}$$

As an instance of this procedure, we can compute from a set of Horn clauses defining Red , a set of Horn clauses defining $\neg Red$; typically Red can be defined by:

$$\begin{array}{ll} Red(s) & \text{for every left hand side } s \text{ of a rule in } \mathcal{R} \\ Red(f(s_1, \dots, s_n)) \Leftarrow Red(s_i) & \text{for every } f \in F \text{ and every } i \end{array}$$

which satisfies the variable occurrence property.

6. Ground Reducibility

Ground reducibility is formally defined on page 943. This predicate is also referred to as Red in the previous section. Decision of ground reducibility implies the decision of consistency with the normal axiomatizations given in sections 5.3 and 5.4. Its role however goes beyond this simple application: first, it is strongly related to sufficient completeness (see definition 2.6). Next, the works which have been devoted to this property introduce some techniques which are used for other purposes. For instance, *test sets* are the basis of the mechanization of induction in SPIKE [Bouhoula and Rusinowitch 1995] and *cover sets* are the basis of the mechanization of induction in (a part of) RRL [Kapur and Zhang 1995]. *Ground normal form languages* and

their recognizers play a role in specification analysis in ReDuX and in the design of new induction techniques [Jouannaud and Bouhoula 1997].

The decidability of ground reducibility was shown originally by D. Plaisted [Plaisted 1985] and several other algorithms have been proposed since (e.g. [Jouannaud and Kounalis 1989, Kapur, Narendran and Zhang 1987, Kapur, Narendran, Rosenkrantz and Zhang 1991, Kounalis 1992, Caron, Coquidé and Dauchet 1993, Comon and Jacquemard 1997]). We sketch here two of them: one is based on *tree automata with constraints* along the lines of [Caron et al. 1993]. The second one is based on test sets, along the lines of [Kapur et al. 1987, Kounalis 1992]. Both of these methods have some feed back on inductive proofs since they yield an explicit induction scheme.

The complexity of Plaisted's decision algorithm is very high (a tower of 7 exponentials). The other algorithms reduce this upper bound. In the general case, the problem, including the computation of a "counter-example" in case of non-ground reducibility, is at least doubly exponential. When we stick to the decision problem, it is EXPTIME-complete [Comon and Jacquemard 1997].

6.1. Automata techniques

In [Caron et al. 1993], the authors show a quite general result. Given a term t , let encomp_t be a unary predicate symbol which is interpreted as the set $[\text{encomp}_t]$ of terms encompassing t .

6.1. THEOREM ([Caron et al. 1993]). *Given n terms t_1, \dots, t_n , the first-order theory of $\text{encomp}_{t_1}, \dots, \text{encomp}_{t_n}$ is decidable.*

On the other hand, ground reducibility of t w.r.t. R can be expressed by a particular formula in this theory: assuming that l_1, \dots, l_n are the left hand sides of R , t is ground reducible iff

$$\forall x. \text{encomp}_t(x) \Rightarrow (\text{encomp}_{l_1}(x) \vee \dots \vee \text{encomp}_{l_n}(x))$$

is valid in the above interpretation.

The proof of this property relies, as in Büchi's theorem, on a correspondence between formulas and automata: the authors design a kind of tree automata such that $[\text{encomp}_t]$ is accepted by such an automaton. Then the closure properties of the corresponding class of languages and the decidability of emptiness leads to the desired property.

It is out of the scope of this chapter to give all details of this result. Nevertheless, we are going to give a flavour of a more restricted result, sufficient for ground reducibility, which has some consequences for inductive proofs. We first define *automata with disequality constraints*. They generalize classical (bottom-up) tree automata by allowing them to check disequalities between subtrees.

6.2. DEFINITION. An *automaton with disequality constraints* (over a given finite alphabet F of function symbols) consists of a finite set of states Q , a subset Q_f

of *final states* and a *transition relation*, which is a finite set of constrained rewrite rules of the form

$$f(q_1(x_1), \dots, q_n(x_n)) \xrightarrow{c} q(f(x_1, \dots, x_n))$$

where q_1, \dots, q_n, q are states and c is a conjunction of *disequality constraints* written $p \neq q$ where p, q are strings of natural numbers. When the conjunction is empty, c is written \top or simply omitted.

When every constraint c is \top , we get a classical bottom-up tree automaton (see e.g. [Comon, Dauchet, Gilleron, Lugiez, Tison and Tommasi 1997]).

The disequality constraints can be viewed as unary predicate symbols and are interpreted as follows: $t \models p \neq q$ iff p, q are positions of t and $t|_p \neq t|_q$. A ground term $t \in T(F \cup Q)$ rewrites to u using the constrained rule $f(q_1(x_1), \dots, q_n(x_n)) \xrightarrow{c} q(f(x_1, \dots, x_n))$ at position p_0 if t rewrites to u (at position p_0) using the unconstrained rule and, moreover, the subterm of t at position p_0 satisfies c . The reduction relation associated with a production rule P is written (as for rewriting) \xrightarrow{P} . The variables x_1, \dots, x_n are always irrelevant when displaying P , hence they are not reported.

6.3. DEFINITION. A ground term t is *accepted* by the automaton (Q, Q_f, P) if $t \xrightarrow{P}^* q(t)$ for some $q \in Q_f$. The *language* $L(A)$ accepted by an automaton A is the set of ground terms that are accepted by A .

6.4. EXAMPLE. Let $F = \{0, s, +\}$, $Q = Q_f = \{q_0, q_s, q_+\}$ and

$$P = \begin{cases} 0 & \rightarrow & q_0 \\ s(q_0) & \rightarrow & q_s \\ s(q_s) & \rightarrow & q_s \\ s(q_+) & \rightarrow & q_+ \\ q_s + q_s & \xrightarrow{1 \neq 2} & q_+ \end{cases}$$

The term $s(0) + s(s(0))$ is accepted by the automaton ($s(s(0))$ and $s(0)$ are accepted in state q_s and the last rule can be applied at the root since $s(s(0)) \neq s(0)$), whereas $s(0) + s(0)$ or $(s(0) + s(s(0))) + s(0)$ are not accepted.

These automata may also be non-deterministic (which is not shown by the example).

Now, the main interest of such automata lies in the two properties given as theorems 6.5 and 6.8.

6.5. THEOREM. *For any rewrite system R , there is an automaton with disequality constraints A which accepts the set of irreducible ground terms.*

6.6. EXAMPLE. The automaton of example 6.4 accepts all ground terms that are irreducible w.r.t. the rewrite system

$$\mathcal{R} = \left\{ \begin{array}{lll} 0 + x & \rightarrow & x \\ x + 0 & \rightarrow & x \\ x + x & \rightarrow & 0 \\ (x + y) + z & \rightarrow & x + (y + z) \\ s(x) + (y + z) & \rightarrow & s(x + (y + z)) \end{array} \right.$$

6.7. EXAMPLE. Let $R_2 = \{x + x \rightarrow 0; 0 + x \rightarrow x; x + 0 \rightarrow x; s(s(0)) \rightarrow 0\}$, and the alphabet be $\{0, s, +\}$. The automaton which recognizes the set of ground normal forms is given by:

$$Q = Q_f = \{q_0, q_{x+y}, q_{s(x)}\}$$

0	\rightarrow	q_0	$s(q_0)$	\rightarrow	$q_{s(x)}$
$s(q_{x+y})$	\rightarrow	$q_{s(x)}$	$s(q_{s(x)})$	\rightarrow	$q_{s(x)}$
$q_{s(x)} + q_{s(x)}$	$\xrightarrow{1 \neq 2}$	q_{x+y}	$q_{s(x)} + q_{x+y}$	\rightarrow	q_{x+y}
$q_{x+y} + q_{s(x)}$	\rightarrow	q_{x+y}	$q_{x+y} + q_{x+y}$	$\xrightarrow{1 \neq 2}$	q_{x+y}

if we do not consider the rules yielding q_r . Moreover, removing the non-accessible states, only two rules remain:

$$0 \rightarrow q_0 \qquad s(q_0) \rightarrow q_{s(x)}$$

6.8. THEOREM. *Given an automaton with disequality constraints \mathcal{A} , it is decidable whether $L(\mathcal{A})$ is empty, resp. finite.*

This theorem is a generalization of the classical emptiness decision problem of finite tree automata.

In general the algorithm is also more complex: it is polynomial in the number of states and exponential in the size of the constraints.

Now, let us show how this can be applied to ground reducibility.

6.9. PROPOSITION. *The set of ground instances of a linear term t is accepted by a finite tree automaton.*

Hence, in the linear case, it is easy to decide ground reducibility: compute the automaton \mathcal{A}_{NF} accepting the set of irreducible ground terms. Compute the automaton \mathcal{A}_t which accepts the set of ground instances of t . Intersect the two automata (this can be done in polynomial time using standard techniques, see e.g. [Comon et al. 1997]). Finally check the emptiness of the resulting automaton. Hence, once the automaton \mathcal{A}_{NF} has been computed, we have:

6.10. PROPOSITION. *Ground reducibility of a linear term w.r.t. a given left linear rewrite system can be decided in polynomial time.*

Note however that the problem becomes EXPTIME-complete if \mathcal{A}_{NF} is not given [Kapur et al. 1991].

In the case of non-linear terms, the problem can also be reduced to emptiness decision of automata with disequality constraints. In any case, it can be solved in deterministic exponential time [Comon and Jacquemard 1997] and that is the best we can do [Kapur et al. 1991].

6.2. Test sets

The basic idea of test sets is very simple: find a finite set of terms TS such that t is ground reducible iff all its instances obtained by replacing its variables by a term in TS are reducible.

Such test sets TS always exist (see [Kapur et al. 1987]), but they may be very large. How is it possible to compute the test sets? The naive computation is quite simple. We build a tree (called a *reducibility tree*). We call i -instance of a term t any ground instance $t\sigma$ of t such that, for every variable x of t , $x\sigma$ is irreducible.

Start with a tree which contains only one node (the root, which is also a leaf), labeled with $t = x$ (a variable). Then we repeat the following steps:

1. Select a leaf of the tree whose label t has reducible i -instances. (i.e. one of its non-variable subterms is unifiable with a left hand side of a rule). If there is no such leaf, then stop.
2. Select a variable x of t (t is not ground because none of the leaves are reducible). Consider the set $S(x, t)$ obtained by splitting x over the signature. More precisely, $S(x, t) = \{t\{x \rightarrow f(\vec{y})\} \mid f \in F\}$. Let $S'(x, t)$ be the subset of irreducible terms in $S(x, t)$.
3. If $S'(x, t)$ is empty, then remove the leaf labeled with t from the tree and remove the parents which have no more child by this transformation.
4. If $S'(x, t)$ is not empty, then add to the node labeled with t as many children as there are elements in $S'(x, t)$ and label them with the elements of $S'(x, t)$.

A test set is given by the leaves of the reducibility tree.

Of course, this simple way of proceeding is not terminating in general. It is however possible to give a bound B on the depth of the terms labeling a node of the reducibility tree. If, in the above algorithm, it is not possible to add a node labeled with a term of size smaller than B , then the computation stops.

An invariant of this computation is that the set of irreducible ground terms is always contained in the set of all i -instances of the leaves.

The bound B is really huge (from 5 to 7 exponentials depending on the version). It is however possible to restrict the computation in various ways. One of the main remark is the following: if, at some computation step, all leaves of a subtree labeled with t are labeled with terms strictly containing some instance of t then we can remove the whole subtree labeled with t . Indeed, assume that $t\sigma$ is a minimal (w.r.t.

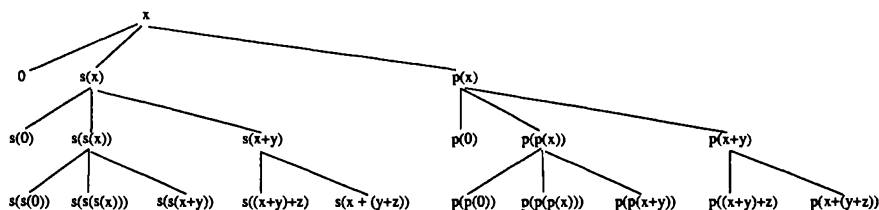
size) irreducible ground instance of t , then it should be an instance of some leaf of the subtree labeled with t , say $t\sigma \equiv u\theta$. However, by hypothesis, there is some strict subterm of u containing an instance of t . This means that there is an irreducible ground instance of t which is strictly smaller than $t\sigma$. Hence a contradiction.

6.11. EXAMPLE. Let us go back again to example 1.1. In order to compute a test set, we start with the variable x . Then split x into 0, $s(x)$ and $x + y$. The two first terms are not unifiable with a left hand side and are thus leaves of the reducibility tree. Remains $x + y$, which is split (say, according to the first variable). We get three terms $0 + y$, which is reducible, $s(x) + y$ which is reducible, and $(x + z) + y$. Then all descendants of $(x + y)$ contain an instance of $x + y$: this node can be removed.

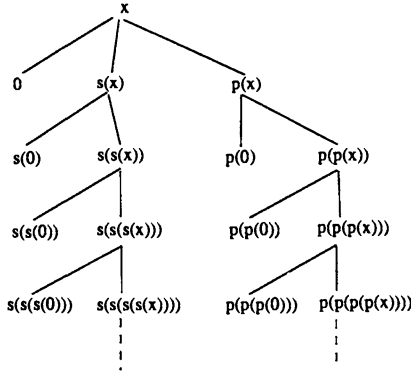
The reducibility tree now only contain one node and two leaves labeled respectively by 0 and $s(x)$. We got a test set containing two elements.

Test sets correspond to induction schemes and test sets computations can be seen as computing an induction scheme which was not explicit. There are however more complicated situations. Let us give a (still quite simple) example.

6.12. EXAMPLE. Consider example 2.7. Starting the computation of the reducibility tree, we get first 0, $s(x)$, $p(x)$, $x + y$. 0 is irreducible and need not be further considered. The three other leaves are however unifiable with left hand sides. Thus, we have to go on adding new leaves; we get the following tree (after some steps):



The node labeled with $x + y$ has been removed according to the criterion: all its descendants contained some instance of it. Then, actually, it is possible to remove in the tree all terms containing $x + y$ (removing a node means that we already proved its ground reducibility). Now, we have the tree:



whose depth depend on the bound B .

It should be clear from example 6.12 that the test set is not “optimal”. To clarify this point and have some feedback on the inductionless induction method, let us study the ground normal form languages.

6.3. Ground normal form languages

In the approaches of the last two sections, the aim was actually to find a description of the irreducible ground terms. In the first case we got a description via tree automata, while in the test set approach, we got a description using a set of instances of a test set.

Actually the test set method does not lead to a real description of irreducible ground terms but to an approximation which is sufficient for ground reducibility purposes only. That is why we stick now to the automata approach.

Assume we have an automaton \mathcal{A} which accepts the set of irreducible ground terms and such that none of the states is useless (i.e. the corresponding language is not empty).

So far, we didn’t consider type information. But it is of course possible to generalize the inductionless induction method to include typing information. More precisely, we may, instead of pure equational logic, consider a logic in which variables are constrained to range over certain tree languages called *sorts*.

6.13. EXAMPLE. Let us define *bool* and *int* as the tree languages accepted in the corresponding states q_{bool} and q_{int} of the automaton whose rules are

$$\begin{array}{ll}
 0 & \rightarrow q_{int} \\
 p(q_{int}) & \rightarrow q_{int} \\
 E(q_{int}) & \rightarrow q_{bool} \\
 false & \rightarrow q_{bool}
 \end{array}
 \qquad
 \begin{array}{ll}
 s(q_{int}) & \rightarrow q_{int} \\
 q_{int} + q_{int} & \rightarrow q_{int} \\
 true & \rightarrow q_{bool}
 \end{array}$$

We will actually write *int* instead of q_{int} and *bool* instead of q_{bool} .

Then the following rules could be a specification of the function symbols:

$$\left\{ \begin{array}{lll} s(p(x)) & \rightarrow & x \\ p(s(x)) & \rightarrow & x \\ 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x + y) \\ p(x) + y & \rightarrow & p(x + y) \\ E(0) & \rightarrow & \text{true} \\ E(s(0)) & \rightarrow & \text{false} \\ E(s(s(x))) & \rightarrow & E(x) \end{array} \quad \begin{array}{l} | \\ | \\ | \\ | \\ | \\ | \\ | \\ | \end{array} \quad \begin{array}{l} x \in \text{int} \\ x \in \text{int} \\ x \in \text{int} \\ x, y \in \text{int} \\ x, y \in \text{int} \\ \\ x \in \text{int} \end{array} \right.$$

Deduction processes may be affected in general by these extensions. But it would lead us too far to consider again a proof system in this context. For the sake of simplicity, we will assume that the sort constraints always satisfy conditions under which exactly the same deduction rules as before are sound. (For example, the *sort decreasingness* property is a sufficient condition; see e.g. [Waldmann 1992]).

Now, introducing new sorts, corresponding to the states of the automaton \mathcal{A} , we may compute from the original specification another specification containing new sorts and new function symbols and which is equivalent to the original one, with additional properties. Here, by “equivalent” we mean that equalities built on the original signature are inductive consequences of the original set of axioms iff they are inductive consequences of the new set of axioms.

Let us illustrate the transformation on an example.

6.14. EXAMPLE. Consider the specification of example 2.7. It is not difficult to see that the automaton accepting the irreducible ground terms computed by the method described in section 6.1 is given by the transition rules:

$$\begin{array}{lll} 0 & \rightarrow & q_0 \\ s(q_{s(x)}) & \rightarrow & q_{s(x)} \\ p(q_{p(x)}) & \rightarrow & q_{p(x)} \end{array} \quad \begin{array}{lll} s(q_0) & \rightarrow & q_{s(x)} \\ p(q_0) & \rightarrow & q_{p(x)} \end{array}$$

We may now call q_0 *zero*, $q_{s(x)}$ *pos* and $q_{p(x)}$ *neg*. We also add a sort *int* containing all terms. We introduce two new function symbols s' and p' and claim that the following specification is equivalent to the original one.

$$\begin{array}{lll} 0 & \rightarrow & \text{zero} \\ s'(\text{zero}) & \rightarrow & \text{pos} \\ s'(\text{pos}) & \rightarrow & \text{pos} \\ p'(\text{zero}) & \rightarrow & \text{neg} \end{array} \quad \begin{array}{lll} s(\text{int}) & \rightarrow & \text{int} \\ p(\text{int}) & \rightarrow & \text{int} \\ \text{int} + \text{int} & \rightarrow & \text{int} \\ p'(\text{neg}) & \rightarrow & \text{neg} \end{array}$$

assuming moreover for sake of simplicity the ϵ -transitions $\text{zero} \rightarrow \text{int}$, $\text{neg} \rightarrow \text{int}$ and $\text{pos} \rightarrow \text{int}$. And the following set of axioms in addition to the original one:

$$\left\{ \begin{array}{ll|l} s(x) & \rightarrow & s'(x) & | & x \in pos \vee x \in zero \\ p(x) & \rightarrow & p'(x) & | & x \in neg \vee x \in zero \\ s(p'(x)) & \rightarrow & x & | & x \in neg \vee x \in zero \\ p(s'(x)) & \rightarrow & x & | & x \in pos \vee x \in zero \\ s'(x) + y & \rightarrow & s(x + y) & | & (x \in pos \vee x \in zero) \wedge y \in int \\ p'(x) + y & \rightarrow & p(x + y) & | & (x \in neg \vee x \in zero) \wedge y \in int \end{array} \right.$$

How to compute this new set of axioms is beyond the scope of these notes (see [Comon 1989] instead).

Now, the main property of this new specification is that $\{0, s', p'\}$ is a set of free constructors! (We leave the proof as an exercise).

Hence it is possible to compute an equivalent (typed) specification for which there are now free constructors. This is not really surprising: the automaton provides the induction schemes that have to be followed in proving inductive properties.

But we cannot conclude directly that only the free constructor case has to be considered. Indeed, when the rewrite systems contain non-linear left hand sides, the typing rules may involve disequality constraints. What equational deduction become in such typed systems is unknown.

6.4. Sufficient Completeness

Sufficient completeness is used in areas other than automated deduction, for example to check that a function defined by pattern matching is indeed completely defined. We have also seen that in some situations, it is important to decide whether some subset of function symbols is a set of constructors.

Actually, sufficient completeness is strongly related to ground reducibility and that is why we add some words on this topic here. First, this property is undecidable (this was first shown in [Guttag and Horning 1978]), even in restricted situations:

6.15. PROPOSITION ([Kapur et al. 1987]). *It is undecidable whether a subset C of F is a set of constructors w.r.t. E , even when E is a finite convergent string rewrite system.*

But this can be recovered as follows:

6.16. PROPOSITION ([Jouannaud and Kounalis 1989]). *Let \mathcal{R} be a convergent rewrite system for which normal forms of terms in $T(C)$ are in $T(C)$. Then C is a set of constructors iff, for each $f \in F \setminus C$, $f(x_1, \dots, x_n)$ is ground reducible.*

Hence techniques that have been developed for ground reducibility apply for sufficient completeness as well: there is no need of further developments.

6.5. *Limitations*

The decidability of ground reducibility implies the decidability of the consistency of a conjecture with a normal axiomatization for the procedures described in sections 5.3 and 5.4, but what about other normal axiomatizations?

Generalizing the construction of section 5.5 it is possible to build normal axiomatizations in several other situations, in particular when E is a finite set of Horn clauses with equality, or when E is a rewrite system modulo associativity and commutativity or even when E is a constrained rewrite system. However, the reducibility predicate is no longer recursive: ground reducibility is undecidable in the case of conditional rewrite systems, in the presence of associative-commutative symbols [Kapur et al. 1991] and for ordered rewriting [Comon, Narendran, Nieuwenhuis and Rusinowitch 1998]. Hence, in such situations, we need to use again a first-order theorem prover which is complete for refutation.

7. A comparison between inductive proofs and proofs by consistency

I express in this section personal opinions about the relationship between explicit (or implicit) induction and proof by consistency as described in this chapter.

Is the proof by consistency method more powerful? Using an induction hypothesis on smaller instances than the instance we are currently considering is a particular case of the redundancy criterion which was defined in section 4. Hence, in principle, the inductive saturation strategies will terminate as soon as an explicit induction (using the same ordering) terminates. There are examples where the saturation process terminates whereas the explicit induction fails. (That is because instances of the original conjecture need not be considered independently). As far as fully automated saturation is concerned, the proof by consistency method, at least in its first phase, is more powerful. It will succeed in all situations where the other method succeeds. Moreover, proof by consistency is a refutationally complete proof procedure. (This can also be obtained from an explicit induction procedure, as it is always possible to enumerate potential counter-examples. But the proof by consistency entirely relies on a as efficient as possible search for such counter-examples). Non-free constructors which cause additional problems in explicit induction techniques are irrelevant problems for the inductive saturation strategy. Finally, the proof by consistency method which was described here is also well-suited for proofs in \mathcal{I}_E , i.e. in the least fixed point of a set of Horn clauses.

Are explicit inductive proofs more powerful? We should not draw conclusions too quickly from the previous paragraph. First, there is a second phase in the proof by consistency approach: we need to prove the consistency of each generated clause with \mathcal{A} . Though this phase is decidable in the equational case, it is EXPTIME-complete. In other cases, there is no precise evaluation of the price to pay for this step. Then, we may also have a different point of view; in the proof by consistency approach, roughly, the induction ordering is not free:

it is the ordering which is used for the saturation of the axiom set.⁴

Finally, is “power” relevant in the context of inductive proofs? In most situations described in Boyer and Moore book [Boyer and Moore 1979], the choice of relevant generalizations is crucial. This is not addressed at all in this chapter. The choice of an appropriate ordering is not addressed either. The main difficulties remain the same in both approaches.

Acknowledgments

I want to thank B. Gramlich, R. Nieuwenhuis, D. Sannella, D. McAllester, L. Fribourg and M. Rusinowitch for their comments on early versions of this chapter.

Bibliography

- BAADER F. AND SNYDER W. [2001], Unification theory, in A. Robinson and A. Voronkov, eds, ‘Handbook of Automated Reasoning’, Vol. I, Elsevier Science, chapter 8, pp. 445–532.
- BACHMAIR L. [1988], Proof by consistency in equational theories, in ‘Proceedings of the Third Symposium on Logic in Computer Science’, Edinburgh, Scotland, pp. 228–233.
- BACHMAIR L. [1991], *Canonical Equational Proofs*, Birkhäuser, Boston.
- BACHMAIR L. AND DERSHOWITZ N. [1994], ‘Equational inference, canonical proofs and proof orderings’, *Journal of the ACM* 41(2), 236–276.
- BACHMAIR L. AND GANZINGER H. [1994], ‘Rewrite-based equational theorem proving with selection and simplification’, *Journal of Logic and Computation* 4(3), 217–247.
- BACHMAIR L. AND GANZINGER H. [2001], Resolution theorem proving, in A. Robinson and A. Voronkov, eds, ‘Handbook of Automated Reasoning’, Vol. I, Elsevier Science, chapter 2, pp. 19–99.
- BOUHOULA A. AND RUSINOWITCH M. [1995], ‘Implicit induction in conditional theories’, *Journal of Automated Reasoning* 14(2), 189–235.
- BOYER R. S. AND MOORE J. S. [1979], *A computational logic*, ACM Monograph Series, Academic Press.
- BUNDY A. [2001], The automation of proof by mathematical induction, in A. Robinson and A. Voronkov, eds, ‘Handbook of Automated Reasoning’, Vol. I, Elsevier Science, chapter 13, pp. 845–911.
- CARON A.-C., COQUIDÉ J.-L. AND DAUCHET M. [1993], Encompassment properties and automata with constraints, in C. Kirchner, ed., ‘5th International Conference on Rewriting Techniques and Applications’, Vol. 690 of *Lecture Notes in Computer Science*, Springer-Verlag, Montreal, Canada.
- COMON H. [1989], Inductive proofs by specifications transformation, in ‘Proc. 3rd Rewriting Techniques and Applications, Chapel Hill, LNCS 355’, Springer-Verlag, pp. 76–91.
- COMON H. [1991], Disunification: a survey, in J.-L. Lassez and G. Plotkin, eds, ‘Computational Logic: Essays in Honor of Alan Robinson’, MIT Press.
- COMON H., DAUCHET M., GILLERON R., LUGIEZ D., TISON S. AND TOMMASI M. [1997], Tree automata techniques and applications. A preliminary version of this unpublished book is available on <http://l3ux02.univ-lille3.fr/tata> .

⁴It is claimed in [Kapur and Zhang n.d.] that a weakness of the proof by consistency approach is the need of a saturated set of axioms. I do not agree with this remark. See e.g. [Comon and Nieuwenhuis 1998] for more details on this.

- COMON H. AND JACQUEMARD F. [1997], Ground reducibility is EXPTIME-complete, in 'Proc. IEEE Symp. on Logic in Computer Science', IEEE Comp. Soc. Press, Warsaw.
- COMON H., NARENDRA P., NIEUWENHUIS R. AND RUSINOWITCH M. [1998], Decision problems in ordered rewriting, in 'Proc. IEEE Symp. Logic in Computer Science', Indianapolis.
- COMON H. AND NIEUWENHUIS R. [1998], Induction = i-axiomatization + first-order consistency, Research Report LSV-98-9, LSV. <http://www.lsv.ens-cachan.fr/Publis/RAPPORTS.LSV/rr-lsv-1998-9.rr.ps>.
- DERSHOWITZ N. [1982], Applications of the Knuth-Bendix completion procedure, in 'Proceedings of the Seminaire d'Informatique Theorique', Paris, France, pp. 95-111.
- DERSHOWITZ N. [1989], Completion and its applications, in H. Aït-Kaci and M. Nivat, eds, 'Resolution of Equations in Algebraic Structures', Vol. 2: Rewriting Techniques, Academic Press, New York, chapter 2, pp. 31-86.
- DERSHOWITZ N. [1991], Canonical sets of Horn clauses, in J. L. Albert, B. Monien and M. R. Artalejo, eds, 'Proceedings of the Eighteenth International Colloquium on Automata, Languages and Programming (Madrid, Spain)', Vol. 510 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 267-278.
- DERSHOWITZ N. AND JOUANNAUD J.-P. [1990], Rewrite systems, in J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B, North-Holland, pp. 243-309.
- DERSHOWITZ N. AND PLAISTED D. [2001], Rewriting, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 9, pp. 533-608.
- FRIBOURG L. [1984], A narrowing procedure for theories with constructors, in R. Shostak, ed., 'Proc. 7th Int. Conf. on Automated Deduction', Vol. 170 of *Lecture Notes in Computer Science*, Springer-Verlag, Napa, CA., pp. 259-281.
- FRIBOURG L. [1986], A strong restriction of the inductive completion procedure, in 'Proc. 13th ICALP, Rennes, LNCS 226', Springer-Verlag, pp. 105-115.
- FRIBOURG L. [1989], 'A strong restriction of the inductive completion procedure', *Journal of Symbolic Computation* 8, 253-276.
- GANZINGER H. AND STUBER J. [1992], Inductive theorem proving by consistency for first-order clauses, in 'Proc. CTRS'92', number 656 in 'LNCS', pp. 226-241.
- GOGUEN J. A. [1980], How to prove algebraic inductive hypotheses without induction, with applications to the correctness of data type implementations, in 'Proceedings of the Fifth International Conference on Automated Deduction (Les Arcs, France)', Vol. 87 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 356-373.
- GRAMLICH B. [1989], Inductive theorem proving using refined unfailing completion techniques, SEKI Report SR-89-14, Fachbereich Informatik, Universität Kaiserslautern.
- GRAMLICH B. [1990a], Completion based inductive theorem proving: A case study in verifying sorting algorithms, SEKI Report SR-90-04, Fachbereich Informatik, Universität Kaiserslautern.
- GRAMLICH B. [1990b], Completion based inductive theorem proving: An abstract framework and its applications, in L. Aiello, ed., 'Proc. 9th European Conf. on Artificial Intelligence', Pitman Publishing, London, pp. 314-319.
- GRAMLICH B. [1990c], UNICOM: a refined completion based inductive theorem prover, in 'Proc. 10th Int. Conf. on Automated Deduction (CADE 90)', Vol. 449 of *LNAI*, Kaiserslautern, pp. 655-656.
- GRAMLICH B. AND LINDNER W. [1991], A guide to UNICOM, an inductive theorem prover based on rewriting and completion techniques, SEKI-Report SR-91-17, Fachbereich Informatik, Universität Kaiserslautern.
- GUTTAG J. V. AND HORNING J. J. [1978], 'The algebraic specification of abstract data types', *Acta Informatica* 10, 27-52.
- HUET G. AND HULLOT J.-M. [1982], 'Proofs by induction in equational theories with constructors', *Journal of Computer and System Sciences* 25(2).

- JOUANNAUD J.-P. AND BOUHOULA A. [1997], Automata-driven automated induction, in 'Twelfth Annual IEEE Symposium on Logic in Computer Science', IEEE Comp. Soc. Press, Warsaw, Poland.
- JOUANNAUD J.-P. AND KOUNALIS E. [1986], Automatic proofs by induction in equational theories without constructors, in 'Proceedings of the Symposium on Logic in Computer Science', Cambridge, MA, pp. 358–366.
- JOUANNAUD J.-P. AND KOUNALIS E. [1989], 'Automatic proofs by induction in theories without constructors', *Information and Computation* **82**(1).
- KAPUR D. AND MUSSER D. [1987], 'Proof by consistency', *Artificial Intelligence* **31**(2).
- KAPUR D., NARENDRA P., ROSENKRANTZ D. AND ZHANG H. [1991], 'Sufficient completeness, ground reducibility and their complexity', *Acta Informatica* **28**, 311–350.
- KAPUR D., NARENDRA P. AND ZHANG H. [1987], 'On sufficient completeness and related properties of term rewriting systems', *Acta Informatica* **24**(4), 395–415.
- KAPUR D. AND ZHANG H. [1995], 'An overview of the rewrite rule laboratory', *Journal of Mathematics of Computation*.
- KAPUR D. AND ZHANG H. [n.d.], Automating induction: Explicit vs inductionless. Unpublished draft.
- KOUNALIS E. [1992], 'Testing for the ground (co)-reducibility in term rewriting systems', *Theoretical Computer Science* **106**(1), 87–117.
- KÜCHLIN W. [1989], Inductive completion by ground proof transformation, in H. Ait-Kaci and M. Nivat, eds, 'Resolution of Equations in Algebraic Structures', Vol. 2: Rewriting Techniques, Academic Press, New York, pp. 211–244.
- LANKFORD D. S. [1981], A simple explanation of inductionless induction, Technical Report MTP-14, Mathematics Department, Louisiana Tech. Univ.
- MUSSER D. R. [1980], On proving inductive properties of abstract data types, in 'Proceedings of the Seventh ACM Symposium on Principles of Programming Languages', Las Vegas, NV, pp. 154–162.
- NIEUWENHUIS R. AND RUBIO A. [1995], 'Theorem proving with ordering and equality constrained clauses', *Journal of Symbolic Computation* **19**(4), 321–351.
- NIEUWENHUIS R. AND RUBIO A. [2001], Paramodulation-based theorem proving, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 7, pp. 371–443.
- PLAISTED D. [1985], 'Semantic confluence tests and completion methods', *Information and Control* **65**, 182–215.
- REDDY U. S. [1990], Term rewriting induction, in M. Stickel, ed., 'Proceedings of the Tenth International Conference on Automated Deduction (Kaiserslautern, West Germany)', Vol. 449 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 162–177.
- SLAGLE J. R. [1974], 'Automated theorem-proving for theories with simplifiers, commutativity, and associativity', *J. of the Association for Computing Machinery* **21**(4), 622–642.
- WALDMANN U. [1992], 'Semantics of order-sorted specifications', *Theoretical Computer Science* **94**, 1–35.
- ZHANG H. [1988], Reduction, Superposition and Induction: Automated Reasoning in Equational Logic, PhD thesis, Rensselaer Polytechnic Institute, Troy, N.Y.

Index

Symbols

$=_E$	920
C^{\leftarrow}	928
<i>Red</i>	948
$T(F)$	919
$T(F, X)$	919
$\downarrow_{\mathcal{R}}$	923
encomp_t	949
$\xleftrightarrow[l=r]{\sigma}$	920
\equiv	919
$\xrightarrow[l \rightarrow r]{\sigma}$	923
Vars	919
$t(p)$	919
$t \mid_p$	919
\mathcal{I}_E	921

A

arity	919
associative commutative symbols	938, 957
atomic formula	920
automaton with disequality constraints	949
axiomatization	915, 925

B

Bachmair	946
----------	-----

C

clause	920
confluent rewrite system	923
Conjecture superposition	931
consequence	920
constructor	922
convergent rewrite system	923
cover set induction	917
critical pair	923

D

defined symbols	922
derivation sequence	930
disequality constraint	950

E

encompassment	920, 949
equality	919
equality resolution	931
equational axiomatization	939
equational deduction	920, 941

F

failing completion procedure	923
------------------------------	-----

fair completion procedure	923
fair derivation sequence	930
free constructor	941
free constructors	922, 941
freely generated	
structure	915
function symbol	919

G

ground normal form languages	954
ground reducibility	919, 943, 944, 948
ground substitution	920
ground term	919

H

Herbrand interpretation	920
Herbrand model	915
Horn clause	920
Huet and Hullot	941

I

I-axiomatization	925
i-instance	952
implicit induction	917
inconsistency	925
induction	916
cover set	917
implicit	917
rewriting	917
structural	922
test set	917, 948
inductive completion	927
inductive consequence	920
inductive saturation strategy	930
inductive theorem	920
inductive theorem proving	915
inductive theory	920
inductively complete position	937
initial model	921
irreducibility predicate	948

J

Jouannaud and Kounalis	943
------------------------	-----

K

Knuth Bendix completion	919, 923, 939, 941
-------------------------	--------------------

L

linear strategy	926, 930, 931, 946
-----------------	--------------------

literal 920

M

minimal Herbrand model ... 915, 916, 921

Musser 938

N

narrowing 931

negation as failure 921

normal axiomatization 933

O

ordered factoring 931

ordered resolution 931

ordered rewriting 957

P

perfect model 925

position 919

positive clause 920

positive literal 920

proof by consistency 916, 917

R

reducibility predicate 948

reducibility tree 952

reduction ordering 928

reduction relation 923

redundancy 919, 928

refutation completeness 930

replacement of equals by equals 920

rewrite relation 923

rewrite rule 923

rewrite system 923

rewriting induction 917

S

saturated set of clauses 929

simplification 929

sort 954

sort decreasingness 955

sort structure 942

structure freely generated 915

substitution 919

subterm 919

sufficient completeness 922, 956

T

term 919

term rewriting 922

terminating rewrite system 923

test set 948, 952

test set induction 917

tree automata with constraints 949

tree automaton 950

U

unfailing completion 947

V

variable 919