Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Harald Ganzinger (Ed.)

Automated Deduction – CADE-16

16th International Conference
on Automated Deduction
Trento, Italy, July 7-10, 1999
Proceedings

Springer

Series Editors
Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editor

Harald Ganzinger
Max-Planck-Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken, Germany
E-mail: Harald.Ganzinger@mpi-sb.mpg.de

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Automated deduction : proceedings / CADE-16, 16th International Conference on Automated Deduction, Trento, Italy, July 7 - 10, 1999, Harald Ganzinger. - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Lecture notes in computer science ; Vol. 1632 : Lecture notes in artificial intelligence)
ISBN 3-540-66222-7

CR Subject Classification (1998): I.2.3, F.4.1, F.3.1

ISBN 3-540-66222-7 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN 10703383 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

This volume contains the papers presented at the Sixteenth International Conference on Automated Deduction (CADE-16), held in Trento, Italy, July 7–10, 1999, and hosted by Istituto Trentino di Cultura – Centro per la ricerca scientifica e tecnologica (ITC-IRST). The year 1999 marks the 25th anniversary of CADE. Since their inception in 1974 at Argonne National Laboratory, the CADE conferences have matured into the major forum for presentation of research in all aspects of automated deduction.

CADE-16 was one of the conferences participating in the 1999 Federated Logic Conference (FLoC). FLoC'99 was the second Federated Logic Conference; the first took place in 1996 and was hosted by DIMACS at Rutgers University, New Brunswick, NJ. The intention of the Federated Logic Conferences is to bring together as a synergetic group several conferences that apply logic to computer science. The other participating conferences in FLoC'99 were the Eleventh International Conference on Computer-Aided Verification (CAV'99), the Fourteenth IEEE Symposium on Logic in Computer Science (LICS'99), and the Tenth Conference on Rewriting Techniques and Applications (RTA-99).

Eighty-three papers were submitted to CADE-16: 67 regular papers and 16 system descriptions. Each of the submissions was reviewed by at least four program committee members, and an electronic program committee meeting was held through the Internet. Of the 83 papers, 21 regular papers and 15 system descriptions were accepted. In addition, this volume contains full papers by two of the four invited speakers, Erich Grädel and Robert Nieuwenhuis, along with an abstract of Tobias Nipkow's invited lecture. Zohar Manna gave an invited talk in a plenary session with CAV.

These proceedings do not cover several important conference events. Four workshops were held on specialized research topics. The fourth automated theorem-proving system competition (CASC-16) was organized by Geoff Sutcliffe. And for the first time an induction system competition was organized by Dieter Hutter.

I would like to thank the many people who have made CADE-16 possible. I am grateful to the following groups and individuals: to the Program Committee and the additional referees for reviewing the papers in a very short time and maintaining the high standard of CADE conferences; to my fellow Trustees for their advice and co-operation; to the FLoC Organizing Committee (Fausto Giunchiglia, Leonid Libkin, Paolo Traverso, Morena Carli, Carola Dori, Alessandro Tuccio, Adolfo Villafiorita, and Moshe Vardi) for organising an excellent and outstanding conference; and last but not least, to Uwe Waldmann, and the other members of my research group at MPI, who helped with the many tasks of the program chair, and helped compensate for his lack of technical expertise in certain Web and text processing tools.

Saarbrücken, May 1999

Harald Ganzinger
CADE-16 Program Chair

Conference Organization

Program Chair

Harald Ganzinger (Saarbrücken)

Assistant to Program Chair

Uwe Waldmann

Program Committee

Franz Baader (Aachen)
Leo Bachmair (Stony Brook)
David Basin (Freiburg)
Alan Bundy (Edinburgh)
Hubert Comon (Cachan)
Gilles Dowek (Rocquencourt)
Harald Ganzinger (Saarbrücken)
Ryuzo Hasegawa (Kyushu)
Jieh Hsiang (Taipei)
Deepak Kapur (Albany)
Michael Kohlhase (Saarbrücken)
Hélène Kirchner (Nancy)
Alexander Leitsch (Vienna)
Reinhold Letz (Munich)
Patrick Lincoln (Menlo Park)
Christopher Lynch (Clarkson)
David McAllester (Florham Park)
William McCune (Argonne)
Robert Nieuwenhuis (Barcelona)
Hans de Nivelle (Amsterdam)
Hans Jürgen Ohlbach (London)
Lawrence Paulson (Cambridge)
Frank Pfenning (Pittsburgh)
David Plaisted (Chapel Hill)
John Slaney (Canberra)
Tanel Tammet (Göteborg)
Andrei Voronkov (Manchester)
Lincoln Wallen (Oxford)
Dongming Wang (Grenoble)

FLoC General Chair

Moshe Vardi

FLoC Publicity Chair

Leonid Libkin

Local Organization

Fausto Giunchiglia (Conference Chair)
Paolo Traverso (CADE Local Arrangements Chair)
Morena Carli, Carola Dori (Secretaries)
Alessandro Tuccio (Treasurer)
Adolfo Villafiorita (National Publicity Chair and Workshops Coordinator)

List of Referees

L. Araujo
A. Armando
T. Arts
M. Baaz
B. Beckert
G. Bella
S. Berardi
M. Bishop
A. Blackwell
A. Bockmayr
M. P. Bonacina
R. Boulton
T. Boy de la Tour
C. E. Brown
R. Caferra
D. Calvanese
A. Colmerauer
S. Colton
S. Cresswell
G. Défourneaux
A. Degtyarev
R. Diaconescu
U. Egly
W. M. Farmer
C. Fermüller
L. Fribourg
H. Fujita
J. Giesl
R. Goré
J. Gow
B. Gramlich
I. Green
M. Hanus
L. Hendriks
M. Hermann
A. Herzig
I. Horrocks
J. Hurd
D. Hutter
K. Inoue
M. Jackson
M. Jaeger
B. Keller
N. Klarlund
K. Korovin
M. Koshimura
J. Levy
C.-J. Liau
M. Lifantsev
G. Lowe
H. Lowe
C. Lutz
H. Mantel
C. Marché
S. Matthews
A. Meier
C. Meyer
P. Mielniczuk
T. Minami
R. Monroy
A. Mycroft
P. Narendran
A. Nonnengart
M. Norrish
Y. Ohta
T. Ohtani
A. Oliart
S. Owre
L. Pacholski
S. Pearson
N. Peltier
B. Pientka
A. Pitts
C. R. Ramakrishnan
A. Riazanov
J. Richardson
C. Ringeissen
A. Rubio
H. Rueß
K. Sakai
G. Salzer
D. Sannella
H. Schellinx
R. Schmidt
S. Schneider
C. Schürmann
N. Shilov
W. Snyder
V. Sofronie-Stokkermans
V. Sorge
M. Staples
J. Stark
R. Statman
G. Struth
D. Syme
P. Thiemann
A. Tiwari
T. E. Uribe
G. Valiente
M. VanInwegen
M. Veanes
K. Vershinin
L. Viganò
L. Vigneron
T. Walsh
K. Watkins
C. Weidenbach
T. Weigert
P. Wickline
A. Wolf
B. Wolff
H. Zhang

Previous CADEs

CADE-1, Argonne National Lab., USA, 1974 (IEEE Trans. on Comp. C-25(8))
CADE-2, Oberwolfach, Germany, 1976
CADE-3, MIT, USA, 1977
CADE-4, University of Texas at Austin, USA, 1979
CADE-5, Les Arcs, France, 1980 (Springer, LNCS 87)
CADE-6, Courant Institute, New York, USA, 1982 (Springer, LNCS 138)
CADE-7, Napa, California, USA, 1984 (Springer, LNCS 170)
CADE-8, University of Oxford, UK, 1986 (Springer, LNCS 230)
CADE-9, Argonne National Laboratory, USA, 1988 (Springer, LNCS 310)
CADE-10, Kaiserslautern, Germany, 1990 (Springer, LNAI 449)
CADE-11, Saratoga Springs, New York, USA, 1992 (Springer, LNAI 607)
CADE-12, Nancy, France, 1994 (Springer, LNAI 814)
CADE-13, Rutgers University, USA, 1996 (Springer, LNAI 1104)
CADE-14, Townsville, Australia, 1997 (Springer, LNAI 1249)
CADE-15, Lindau, Germany, 1998 (Springer, LNAI 1421)

CADE Inc. Trustees

CADE-16 Sponsors

AI∗IA – Associazione Italiana per l'Intelligenza Artificiale
Azienda per la Promozione Turistica di Trento
Azienda per la Promozione Turistica del Trentino
CADE Inc.
CCL II – Constraints in Computational Logic
Comune di Trento
Dipartimento di Informatica e Studi Aziendali, Università di Trento
European Commission
Facoltà di Lettere e Filosofia, Università di Trento
Invisible Site – Communications Technologies
Provincia Autonoma di Trento
Regione Autonoma del Trentino Alto-Adige
Sun Microsystems
Telecom Italia
Università di Trento

Table of Contents

A Dynamic Programming Approach to Categorial Deduction

Philippe de Groote

LORIA UMR n^o 7503 – INRIA, Campus Scientifique, B.P. 239
54506 Vandœuvre lès Nancy Cedex – France
degroote@loria.fr

Abstract. We reduce the provability problem of any formula of the Lambek calculus to some context-free parsing problem. This reduction, which is based on non-commutative proof-net theory, allows us to derive an automatic categorial deduction algorithm akin to the well-known Cocke-Kasami-Younger parsing algorithm.

1 Introduction

Modern categorial grammars [8], which are intended to give a deductive account of grammatical composition, are based on substructural logics whose paradigm is the Lambek calculus [7]. Any parsing problem for a given categorial grammar gives rise to some decision problem in the substructural logic on which the grammar is based. Consequently, to design an efficient parsing algorithm for a categorial grammar amounts to design an efficient decision procedure for some logic akin to the Lambek calculus.

On the other hand, the Lambek calculus, which has been introduced four decades ago, appears *a posteriori* to be a non-commutative fragment of linear logic [4]. Consequently, it is possible to take advantage of Girard's proof-net theory in order to design automatic deduction procedures for the Lambek calculus.

In this paper, we introduce an original correctness criterion for non-commutative proof-nets, from which we derive a dynamic programming algorithm for deciding the provability of a Lambek sequent.

The paper is organised as follows. Section 2 is a short presentation of the Lambek calculus. In Section 3 we introduce an original notion of non-commutative proof-net, and we prove that this notion of proof-net corresponds to an actual notion of proof. In Section 4, we associate to each Lambek sequent a context-free grammar that allows to see the provability problem, for this sequent, as a rewriting problem. This gives rise to the algorithm described in Section 5. Finally, we conclude in Section 6.

It is to be noted that the theory of non-commutative proof-nets appeals to the theory of planar graphs. The formalisation of this theory relies on advanced geometrical concepts whose exposition is out of the scope of this paper. On the other hand, the elementary concepts of planar graph theory that we used (essentially the notion of *face*) are rather intuitive. Consequently, we have decided to illustrate the different concepts by several examples in order not to lose the reader into the formal details.

2 The Lambek Calculus

The Lambek calculus [7], which has been introduced as a logical basis for categorial grammars [8], corresponds exactly to the non-commutative intuitionistic multiplicative fragment of linear logic [4].

Given an alphabet of atomic formulas $\mathcal{A}$, the syntax of the formulas obeys the following grammar:

$$\mathcal{F} \ ::= \ \mathcal{A} \mid \mathcal{F} \bullet \mathcal{F} \mid \mathcal{F} \backslash \mathcal{F} \mid \mathcal{F} / \mathcal{F}$$

where formulas of the form $A \bullet B$ correspond to conjunctions (or products), formulas of the form $A \backslash B$ correspond to direct implications (i.e., A *implies* B), and formulas of the form A/B to retro-implication (i.e., A *is implied by* B).

Then, the deduction relation is specified by means of the following sequent calculus.

$$A \vdash A \quad \text{(ident)} \qquad\qquad \frac{\Gamma \vdash A \quad \Delta_1, A, \Delta_2 \vdash B}{\Delta_1, \Gamma, \Delta_2 \vdash B} \ \text{(cut)}$$

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, A \bullet B, \Delta \vdash C} \ (\bullet \text{ left}) \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \bullet B} \ (\bullet \text{ right})$$

$$\frac{\Gamma \vdash A \quad \Delta_1, B, \Delta_2 \vdash C}{\Delta_1, \Gamma, A \backslash B, \Delta_2 \vdash C} \ (\backslash \text{ left}) \qquad \frac{A, \Gamma \vdash B}{\Gamma \vdash A \backslash B} \ (\backslash \text{ right})$$

$$\frac{\Gamma \vdash A \quad \Delta_1, B, \Delta_2 \vdash C}{\Delta_1, B/A, \Gamma, \Delta_2 \vdash C} \ (/ \text{ left}) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash B/A} \ (/ \text{ right})$$

It is to be noted that the above system does not include any structural rule. In particular, the absence of an exchange rule is responsible for the non-commutativity of the connectives. This, in turn, explains the presence of two different implications.

Example 1. As an illustration, consider the sequent

$$(a/b) \bullet b,\, b \backslash (b \bullet (a \backslash a)) \vdash a,$$

which may be derived as follows:

$$\frac{\dfrac{b \vdash b \qquad \dfrac{\dfrac{b \vdash b \qquad \dfrac{a \vdash a \quad a \vdash a}{a,\, a \backslash a \vdash a}}{a/b,\, b,\, a \backslash a \vdash a}}{a/b,\, b \bullet (a \backslash a) \vdash a}}{a/b,\, b,\, b \backslash (b \bullet (a \backslash a)) \vdash a}}{(a/b) \bullet b,\, b \backslash (b \bullet (a \backslash a)) \vdash a}$$

3 Proof-nets for the Lambek calculus

Proof-nets are concrete structures that allow the proofs of linear logic to be represented geometrically [4]. As we mentioned, the Lambek calculus is a non-commutative fragment of linear logic. It is consequently possible to adapt the notion of proof net to the case of the Lambek calculus [6, 10]. To this end, we first give a translation of the Lambek calculus into (non-commutative) multiplicative linear logic.

The formulas of multiplicative linear logic are built upon a set of atomic propositions $\mathcal{A}$ according to the following grammar:

$$\mathcal{MF} \ ::= \ \mathcal{A} \mid \mathcal{A}^{\perp} \mid \mathcal{MF} \otimes \mathcal{MF} \mid \mathcal{MF} \parr \mathcal{MF}$$

where the unary connective $^{\perp}$ denotes the linear negation, and the binary connectives $\otimes$ (*tensor*) and $\parr$ (*par*) correspond to multiplicative conjunction and disjunction respectively .

Definition 1. *The* translation $\mathcal{T}[\![\Gamma \vdash A]\!]$ *of a* Lambek sequent $\Gamma \vdash A$ *into a* sequence *of multiplicative formulas is defined as follows:*

$$\mathcal{T}[\![\Gamma \vdash A]\!] = \mathcal{T}^{-}[\![\Gamma]\!]^{-}, \mathcal{T}^{+}[\![A]\!]^{+}$$

where:

1. $\mathcal{T}^{+}[\![a]\!]^{+} = a$
2. $\mathcal{T}^{+}[\![A \bullet B]\!]^{+} = \mathcal{T}^{+}[\![B]\!]^{+} \otimes \mathcal{T}^{+}[\![A]\!]^{+}$
3. $\mathcal{T}^{+}[\![A \setminus B]\!]^{+} = \mathcal{T}^{+}[\![B]\!]^{+} \parr \mathcal{T}^{-}[\![A]\!]^{-}$
4. $\mathcal{T}^{+}[\![A / B]\!]^{+} = \mathcal{T}^{-}[\![B]\!]^{-} \parr \mathcal{T}^{+}[\![A]\!]^{+}$
5. $\mathcal{T}^{-}[\![a]\!]^{-} = a^{\perp}$
6. $\mathcal{T}^{-}[\![A \bullet B]\!]^{-} = \mathcal{T}^{-}[\![A]\!]^{-} \parr \mathcal{T}^{-}[\![B]\!]^{-}$
7. $\mathcal{T}^{-}[\![A \setminus B]\!]^{-} = \mathcal{T}^{+}[\![A]\!]^{+} \otimes \mathcal{T}^{-}[\![B]\!]^{-}$
8. $\mathcal{T}^{-}[\![A / B]\!]^{-} = \mathcal{T}^{-}[\![A]\!]^{-} \otimes \mathcal{T}^{+}[\![B]\!]^{+}$

9. $$\mathcal{T}^{-}[\![\Gamma, A]\!]^{-} = \mathcal{T}^{-}[\![\Gamma]\!]^{-}, \mathcal{T}^{-}[\![A]\!]^{-}$$

It is important to note that the translation $\mathcal{T}[\![\Gamma \vdash A]\!]$ does not define a set but a sequence of formulas. This is due to to fact that the order between the formulas of a Lambek sequent is relevant.

Example 2. Consider the sequent of Example 1. Its translation is the following:

$$\mathcal{T}[\![(a/b) \bullet b,\ b \setminus (b \bullet (a \setminus a)) \vdash a]\!] = (a^{\perp} \otimes b) \parr b^{\perp}, b \otimes (b^{\perp} \parr (a \otimes a^{\perp})), a$$

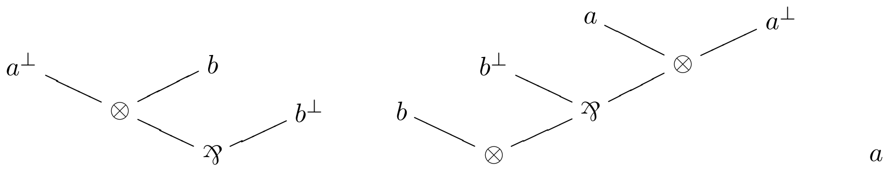
The definition of a multiplicative proof-net procceds as follows. The notion of proof-structure, which corresponds to a class of graphs decorated with formulas, is first defined. These proof-structures are intended to represents proofs. It is not the case, however, that they all correspond to actual proofs. It is therefore necessary to give some further criterion in order to distinguish the *correct* proof-structures, which are called proof-nets, from the other ones.

The notion of multiplicative proof-net may be adapted to the Lambek calculus by stating an additional condition that ensures non-commutativity. We do not follow this approach. The definition that we give is based on a correctness criterion that is intrinsically non-commutative. This new criterion, which has been especially devised in order to prove the correctness of the algorithm of Section 5, is a refinement of a similar criterion due to Fleury [3].

Proof-nets and proof-structure being a sort of graph, we use freely elementary graph-theoretic concepts that can be found in any textbook. In particular, the terminology we adopt is taken from [2]. We also take for granted the notion of parse tree of a formula. The leaves of such a parse tree are decorated with literals (i.e., atomic propositions, or negations of atomic propositions) and its internal nodes are decorated either with the connective $\otimes$ or the connective $\parr$. These internal nodes will be called $\otimes$- and $\parr$-nodes, respectively.

Definition 2. *Let $\Gamma \vdash A$ be a Lambek sequent. The* proof-frame *of $\Gamma \vdash A$ consists of (the sequence of) the parse-trees of the formulas in $\mathcal{T}[\![\Gamma \vdash A]\!]$. The roots of these parse trees are called the* conclusions *(or the conclusive nodes) of the proof-frame.*

Example 3. The proof-frame of the sequent of Example 1 is the following:



Let us associate two *polarised atoms* a^+ and a^- to each atomic formula a, and let Σ_1 be the set of these polarised atoms. The next step in defining a notion of proof-structure for the Lambek calculus consists in introducing the concept of *well-bracketed matching* on a word of Σ_1^*.

Definition 3. *Consider a word $\omega = \omega_1\omega_2 \ldots \omega_n$ of polarised atoms. We define a well-bracketed matching on ω to be a permutation p on the set of ω_i's such that:*

1. *$(\forall i, j \leq n)$ $p(\omega_i) = \omega_j$ implies ($\omega_i = a^+$ and $\omega_j = a^-$) or ($\omega_i = a^-$ and $\omega_j = a^+$), for some atomic formula a.*
2. *$(\forall i, j \leq n)$ $p(\omega_i) = \omega_j$ implies $p(\omega_j) = \omega_i$,*
3. *$(\forall i, j, k, l \in n)$ $p(\omega_i) = \omega_j$, $p(\omega_k) = \omega_l$, and $i < k$ imply $l < j$.*

Conditions 1 and 2, in this definition, formalise the idea that a matching consists in grouping by pairs atoms of opposite polarities. Condition 3 corresponds to a notion of well-bracketed structures by forbidding configurations such as
$\cdots \underset{i}{[} \cdots \underset{k}{(} \cdots \underset{j}{]} \cdots \underset{l}{)} \cdots$

Let $\Gamma \vdash A$ be a Lambek sequent. We write (A/Γ) to denote the sequent (consisting of one formula) obtained by applying Rule (/ right) to $\Gamma \vdash A$ as many times as possible. It is well-known that $\Gamma \vdash A$ is provable if and only if (A/Γ) is. We now associate, to each Lambek sequent, a word of polarised atoms as follows.

Definition 4. *The* word of polarised atoms $\mathcal{V}_1[\![\Gamma \vdash A]\!]$ *associated to a* Lambek sequent $\Gamma \vdash A$ *is defined as* $\mathcal{V}_1[\![\Gamma \vdash A]\!] = \mathcal{W}[\![\mathcal{T}[\![A/\Gamma]\!]]\!]$, *where:*

1. $\mathcal{W}[\![a]\!] = a^+$ *3.* $\mathcal{W}[\![\alpha \parr \beta]\!] = \mathcal{W}[\![\alpha]\!]\,\mathcal{W}[\![\beta]\!]$
2. $\mathcal{W}[\![a^\perp]\!] = a^-$ *4.* $\mathcal{W}[\![\alpha \otimes \beta]\!] = \mathcal{W}[\![\alpha]\!]\,\mathcal{W}[\![\beta]\!]$

Let us interpret a^+ and a^- respectively as a and $a^\perp$, which is consistent with Equations 1 and 2 in the above definition. Then the word associated to a sequent corresponds exactly to the sequence of the leaves of the proof-frame of this sequent.

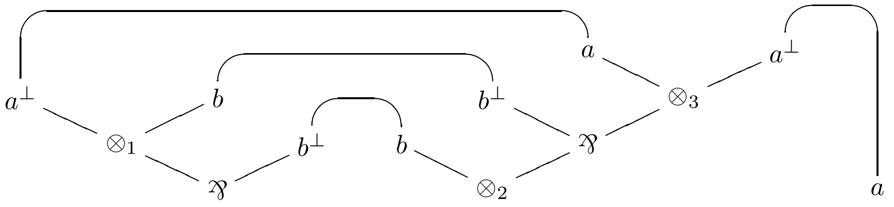
We are now in a position of defining a notion of proof-structure for the Lambek calculus.

Definition 5. *Let $\Gamma \vdash A$ be a Lambek sequent. A proof-structure of $\Gamma \vdash A$ (if any) is a simple decorated graph made of:*

1. *the proof-frame of $\Gamma \vdash A$;*
2. *a perfect matching on the leaves of this proof-frame that corresponds to a well-bracketed matching on $\mathcal{V}_1[\![\Gamma \vdash A]\!]$.*

The conclusions of the proof-structure are defined to be the conclusions of the proof-frame. The edges defining the perfect matching on the leaves of the proof-frame are called the axiom links of the proof-structure.

Example 4. The following graph is a proof-structure for the sequent of Example 1.



In order to state the definition of a proof-net, we need some elementary concepts of planar graph-theory. A graph is said to be planar if it may be represented on a plane in such a way that no two edges cross one another. Such a representation is called a *topological planar graph.* A *face* of a topological planar graph is defined to be a region of the plane bounded by edges in such a way that any two points of this region may be connected by a continuous curve that does not cross any edge.

It is is easy to see that the proof-structures of Definition 5 are planar graphs (this is due to the well-bracketing condition). Consider, for instance, the proof-structure of Example 4. It has three faces: two *bounded faces* that are contained within elementary cycles — respectively, (a, $a^\perp$, $\otimes_1$, b, $b^\perp$, $\parr$, $\otimes_3$) and ($b^\perp$, b, $\otimes_1$, $\parr$, $b^\perp$, b, $\otimes_2$, $\parr$) —, and one *unbounded face* that corresponds to the region of the plane that is "outside of the proof-structure".

While the notion of face is proper to the notion of planar graph, the notion of bounded and unbounded face depends of the particular topological planar representation under consideration. Consequently, from now on, when speaking of a proof-net we mean its *natural topological representation*[1] as illustrated by example 4. This convention allows us to define the *faces of a proof-structure* as the bounded faces of the corresponding topological planar graph.

Let P be a proof-structure, F be a face of P and n be a $\parr$- or a $\otimes$-node of P. We say that F *contains* n if and only if n and its two daughter nodes belong to the boundary of F. Remark that some nodes may belong to the boundaries of two different faces. However, according to the present definition, any node *is contained* in at most one face. Finally, let t and u be two $\otimes$-nodes of P. We say that t is dominated by u (and we write $t \prec u$) if and only if t is the ancestor of a $\parr$-node that is contained in the same face as u.

We now define our notion of proof-net.

Definition 6. *Let $\Gamma \vdash A$ be a Lambek sequent. A proof-net of $\Gamma \vdash A$ (if any) is a proof-structure P of $\Gamma \vdash A$ such that*

1. *each face of P contains exactly one $\parr$-node;*
2. *each $\parr$-node is contained in a face of P;*
3. *the relation of dominance $\prec$ between the $\otimes$-nodes of P is acyclic, i.e., its transitive closure $\prec^+$ is irreflexive.*

Example 5. The proof-structure of example 4 is a proof-net. Indeed, each face contains exactly one $\parr$-node. There are no other $\parr$-nodes. The dominance relation consists simply of $\otimes_2 \prec \otimes_1$, whose transitive closure is clearly acyclic.

In constructing a proof-net, the difficulty consists in guessing an appropriate set of axiom links. A possible solution to this problem is to generate all the possible sets of axiom links and to check for each corresponding proof structure whether the proof net correctness criterion is satisfied or not. A more clever way of proceeding is to construct the set of axiom links incrementally so that the correctness criterion is ensured by construction. This is precisely what our algorithm is doing. Consequently, in order to prove its correctness, we will need a notion of partial proof-net.

Let us define a *module* to be a simple decorated graph made of a proof-frame and a partial perfect matching that obeys the conditions of Definition 5. In other words a module is a proof-structure from which some axiom links have been erased. Now, given a module M, we define an *possibly open face* to be a set of nodes that belong to the boundary of the same face in any proof-structure that contains M as a subgraph. If the boundary of a possibly open face is cyclic, we call it an *actual face.* If it is acyclic, we call it an *open face.*

The notion of possibly open face allows the relation of dominance to be defined on modules. This, in turn, allows us to define a *correct module.* to be a

[1] That is the representation obtained by drawing trees with their root at the bottom, by drawing left and right daughter-nodes respectively on the upper left and upper right of their mother, and by drawing sequences of trees from left to right.

module whose actual faces contain exactly one $\parr$-node, whose open faces contain at most one $\parr$-node, and for which the transitive closure of the dominance relation is acyclic.

We end this section by proving that the proof-nets of definition 6 correspond actually to a notion of proof for the Lambek calculus. In other words, we prove that any Lambek sequent $\Gamma \vdash A$ is provable if and only if there exists a proof-net for it.

Proposition 1. *Let $\Gamma \vdash A$ be a provable Lambek sequent. Then there exists a proof-net whose conclusions are $\mathcal{T}[\![\Gamma \vdash A]\!]$.*

Proof. *The proof consists in a straightforward induction on the sequent calculus derivation.* □

To prove the converse of this proposition, which corresponds to Girard's sequentialisation theorem [4, 5], we establish the following key lemma.

Lemma 1. *Let P be a proof-net that does not contain any conclusive $\parr$-node. If P contains at least one $\otimes$-node then it contains a conclusive $\otimes$-node whose removal splits P into two disconnected proof-nets.*

Proof. *Consider some conclusive $\otimes$-node t_1 of P (there is at least one). If the removal of t_1 disconnects P, we are done. Otherwise, t_1 must be contained in a face of P. Consider the $\parr$-node p that is contained in this face. p cannot be a conclusion of P. Hence it must be the descendant of some conclusive $\otimes$-node t_2, and we have $t_2 \prec t_1$. By iterating this process, which terminates because of the acyclicity of the transitive closure of $\prec$, we eventually find a splitting $\otimes$.* □

Proposition 2. *Let P be a proof-net whose conclusions are $\mathcal{T}[\![\Gamma \vdash A]\!]$. Then the Lambek sequent $\Gamma \vdash A$ is provable.*

Proof. *The proof is done by induction on the number of nodes in P. If P is made of a single axiom link then it corresponds to an axiom $a \vdash a$. If P has at least one conclusive $\parr$-node, the induction is straightforward. Finally, if P does not consist of a single axiom link and does not contain any conclusive $\parr$-node, we apply Lemma 1.* □

Proposition 3. *Any Lambek sequent is provable if and only if there exist a proof-net of it.* □

4 Grammars of axiom links

As we already stressed, to construct a proof-net consists essentially in finding an appropriate set of axiom links. This set must obey three conditions:

A. it must induce a well-bracketing on the leaves of the given proof-frame,
B. it must give rise to a proof-structure whose faces contain exactly one $\parr$,
C. it must induce a dominance relation σ whose transitive closure σ^+ is acyclic.

Given a Lambek sequent $\Gamma \vdash A$, it is easy to show that there exist a set of axiom links satisfying Condition A if and only if

$$\mathcal{V}_1[\![\Gamma \vdash A]\!] \quad \rightarrow^* \quad S \qquad\qquad (*)$$

according to System R_1 that is defined by the following rules:

$$\begin{array}{rcllrcll} T & \rightarrow & S & (1) & a^- \, a^+ & \rightarrow & T & (4) \\ T\,S & \rightarrow & S & (2) & a^+ \, S \, a^- & \rightarrow & T & (5) \\ a^+ \, a^- & \rightarrow & T & (3) & a^- \, S \, a^+ & \rightarrow & T & (6) \end{array}$$

Indeed, the above system corresponds to a context-free grammar whose terminal alphabet is Σ_1 and whose non-terminal alphabet is the set $\{S, T\}$. Then, any rewriting sequence such as $(*)$ induces the well-bracketed matching that matches together two atoms of opposite polarities if and only if they are rewritten at the same time by an instance of Rule (3), (4), (5) or (6).

We now transform, step by step, the above grammar in order to take Conditions B and C into account.

Let us define a new terminal alphabet Σ_2 by associating to each $a^+ \in \Sigma_1$ (respectively, $a^- \in \Sigma_1$) two different symbols, namely, $a^+_{\parr}$, $a^+_{\otimes}$ (respectively, $a^-_{\parr}$, and $a^-_{\otimes}$). Similarly, we consider the non-terminal alphabet $\{S_{\parr}, S_{\otimes}, T_{\parr}, T_{\otimes}\}$. Then, we associate to each Lambek sequent $\Gamma \vdash A$ a word of Σ_2^* as follows:

$$\mathcal{V}_2[\![\Gamma \vdash A]\!] = \mathcal{W}[\![\mathcal{T}[\![A/\Gamma]\!]]\!]_{\parr}$$

where the transformation $\mathcal{W} : \mathcal{MF} \times \{\parr, \otimes\} \rightarrow \Sigma_2^*$ obeys the following equations:

1. $\mathcal{W}[\![a]\!]_{\mathbf{c}} = a^+_{\mathbf{c}}$ 3. $\mathcal{W}[\![\alpha \parr \beta]\!]_{\mathbf{c}} = \mathcal{W}[\![\alpha]\!]_{\parr} \, \mathcal{W}[\![\beta]\!]_{\mathbf{c}}$
2. $\mathcal{W}[\![a^{\perp}]\!]_{\mathbf{c}} = a^-_{\mathbf{c}}$ 4. $\mathcal{W}[\![\alpha \otimes \beta]\!]_{\mathbf{c}} = \mathcal{W}[\![\alpha]\!]_{\otimes} \, \mathcal{W}[\![\beta]\!]_{\mathbf{c}}$

with $\mathbf{c}$ ranging over $\{\parr, \otimes\}$. The intended meaning of this definition is the following: $\mathcal{V}_2[\![\Gamma \vdash A]\!]$ corresponds to the sequence of the leaves of the proof-frame of $\Gamma \vdash A$ with, for each leave, a subscript $\parr$ or $\otimes$ indicating respectively whether the open face immediately on the right of the leave contains or does not contain a $\parr$-node.

Let $\Gamma \vdash A$ be a Lambek sequent. We claim that there exists a proof-structure of $\Gamma \vdash A$ whose each face contains exactly one $\parr$-node if and only if

$$\mathcal{V}_2[\![\Gamma \vdash A]\!] \quad \rightarrow^* \quad S_{\parr} \qquad\qquad (**)$$

according System R_2 that is defined by the following rules:

$$\begin{array}{rcllrcll} T_{\mathbf{c}} & \rightarrow & S_{\mathbf{c}} & (1) & a^-_{\parr} \, a^+_{\mathbf{c}} & \rightarrow & T_{\mathbf{c}} & (6) \\ T_{\otimes} \, S_{\parr} & \rightarrow & S_{\parr} & (2) & a^+_{\parr} \, S_{\otimes} \, a^-_{\mathbf{c}} & \rightarrow & T_{\mathbf{c}} & (7) \\ T_{\parr} \, S_{\otimes} & \rightarrow & S_{\parr} & (3) & a^-_{\parr} \, S_{\otimes} \, a^+_{\mathbf{c}} & \rightarrow & T_{\mathbf{c}} & (8) \\ T_{\otimes} \, S_{\otimes} & \rightarrow & S_{\otimes} & (4) & a^+_{\otimes} \, S_{\parr} \, a^-_{\mathbf{c}} & \rightarrow & T_{\mathbf{c}} & (9) \\ a^+_{\parr} \, a^-_{\mathbf{c}} & \rightarrow & T_{\mathbf{c}} & (5) & a^-_{\otimes} \, S_{\parr} \, a^+_{\mathbf{c}} & \rightarrow & T_{\mathbf{c}} & (10) \end{array}$$

where $\mathbf{c}$ ranges over $\{\parr, \otimes\}$.

The only-if-part of our claim is easy to show. In order to prove the if-part, we must show that the proof-frame of $\Gamma \vdash A$ admits a set of axiom links satisfying both Conditions A and B. The proof that $(**)$ induces a set of axiom links satisfying Condition A is similar to the case of System R_1. The proof that this set of axioms satisfies Condition B is based on the following facts:

- for each (occurrence of a) non-terminal symbol N involved in $(**)$ there exist $\omega_1, \omega_2, \omega_3 \in \Sigma_2^*$ such that: (1) $\mathcal{V}_2[\![\Gamma \vdash A]\!] = \omega_1\omega_2\omega_3$; (2) $\omega_2 \rightarrow^* N$; (3) $\omega_1 N \omega_2 \rightarrow^* S_{⅋}$. consequently, to each (occurrence of a) non-terminal symbol N, one may associate the portion of the proof-frame that corresponds to ω_2, the set of axiom links that is induced by $\omega_2 \rightarrow^* N$, and the module that is made of the proof-frame together with this set of axiom links;
- to each (occurrence of a) non-terminal symbol N involved in $(**)$ corresponds (a portion of) an open face of the associated module; this open face corresponds to the portion associated to N together with the leaf immediately on the right of this portion;
- a non-terminal symbol has $⅋$ as a subscript if and only if the corresponding open face contains a $⅋$-node (this property is preserved by all the rules of R_2);
- Rules (5) to (10) ensure that each open face that is turned into an actual face by adding an axiom link contains a $⅋$-node.

We will illustrate the above facts by examples, but first we transform System R_2 in order to take Condition C into account.

From now on, we distinguish the different occurrences of $\otimes$ within a multiplicative formula by means of indices. For instance, we write $(a \otimes_1 (b \otimes_2 c))$ instead of $(a \otimes (b \otimes c))$. Then we consider an infinite set of constants $\mathcal{K} = \{k_1, k_2, k_3, ...\}$, and we construct the alphabet Σ_3 by associating a new symbol $s[\Gamma]$ to any $s \in \Sigma_2$, $\Gamma \subset \mathcal{K}$. Similarly, we associate four different non-terminal symbols $S_{⅋}$, $S_{\otimes}$, $T_{⅋}$, $T_{\otimes}\}$ to any $\Gamma \cup \mathcal{K}$, $\sigma \subset \mathcal{K} \times \mathcal{K}$, such that the transitive closure σ^+ of the relation σ is acyclic. Then, we associate to each Lambek sequent $\Gamma \vdash A$ a word of Σ_3^* defined as follows:

$$\mathcal{V}_3[\![\Gamma \vdash A]\!] = \mathcal{W}[\![\mathcal{T}[\![A/\Gamma]\!]]\!]_{⅋}\, \emptyset\, \emptyset$$

where the transformation $\mathcal{W} : \mathcal{MF} \times \{⅋, \otimes\} \times 2^{\mathcal{K}} \times 2^{\mathcal{K}\times\mathcal{K}} \rightarrow \Sigma_3^*$ obeys the following equations:

1. $\mathcal{W}[\![a]\!]\mathbf{c}\,\Gamma\,\Delta = a^+_{\mathbf{c}}[\Gamma]$
2. $\mathcal{W}[\![a^\perp]\!]\mathbf{c}\,\Gamma\,\Delta = a^-_{\mathbf{c}}[\Gamma]$
3. $\mathcal{W}[\![\alpha ⅋ \beta]\!]\mathbf{c}\,\Gamma\,\Delta = \mathcal{W}[\![\alpha]\!]_{⅋}\,\Delta\,\Delta\;\mathcal{W}[\![\beta]\!]\mathbf{c}\,\Gamma\,\Delta$
4. $\mathcal{W}[\![\alpha \otimes_i \beta]\!]\mathbf{c}\,\Gamma\,\Delta = \mathcal{W}[\![\alpha]\!]_{\otimes}\,\{\mathrm{k}_i\}\,(\Delta \cup \{\mathrm{k}_i\})\;\mathcal{W}[\![\beta]\!]\mathbf{c}\,\Gamma\,(\Delta \cup \{\mathrm{k}_i\})$

with $\mathbf{c}$ ranging over $\{⅋, \otimes\}$. The interpretation of this definition is the following. The different constants $k_1, k_2, \ldots$ occurring in $\mathcal{V}_3[\![\Gamma \vdash A]\!]$ correspond to the different occurrences of $\otimes$ in $\Gamma \vdash A$. Then a symbol $a^+_{⅋}[\Gamma]$ or $a^-_{⅋}[\Gamma]$ occurring in $\mathcal{V}_3[\![\Gamma \vdash A]\!]$ corresponds to a leaf whose right open face contains one $⅋$-node, and Γ corresponds to the set of $\otimes$-nodes that are ancestors of this $⅋$-node. On the other hand, a symbol $a^+_\otimes[\Gamma]$ or $a^-_\otimes[\Gamma]$ corresponds to a leave whose right open face contains one $\otimes$-node (and, consequently, does not contain a $⅋$-node), then Γ is a singleton corresponding to this $\otimes$-node.

Definition 7. *The rewriting system R_3 is defined as follows:*

$$\begin{array}{rcll}
T_{\mathbf{c}}[\Gamma,\sigma] & \to & S_{\mathbf{c}}[\Gamma,\sigma] & (1)\\
T_\otimes[\Gamma_1,\sigma_1]\,S_⅋[\Gamma_2,\sigma_2] & \to & S_⅋[\Gamma_2,\sigma_1\cup\sigma_2\cup(\Gamma_2\times\Gamma_1)] & (2)\\
 & & \multicolumn{2}{r}{\textit{provided that } (\sigma_1\cup\sigma_2\cup(\Gamma_2\times\Gamma_1))^+ \textit{ is acyclic.}}\\
T_⅋[\Gamma_1,\sigma_1]\,S_\otimes[\Gamma_2,\sigma_2] & \to & S_⅋[\Gamma_1,\sigma_1\cup\sigma_2\cup(\Gamma_1\times\Gamma_2)] & (3)\\
 & & \multicolumn{2}{r}{\textit{provided that } (\sigma_1\cup\sigma_2\cup(\Gamma_1\times\Gamma_2))^+ \textit{ is acyclic.}}\\
T_\otimes[\Gamma_1,\sigma_1]\,S_\otimes[\Gamma_2,\sigma_2] & \to & S_\otimes[\Gamma_1\cup\Gamma_2,\sigma_1\cup\sigma_2] & (4)\\
 & & \multicolumn{2}{r}{\textit{provided that } (\sigma_1\cup\sigma_2)^+ \textit{ is acyclic.}}\\
a^+_⅋[\Gamma_1]\,a^-_{\mathbf{c}}[\Gamma_2] & \to & T_{\mathbf{c}}[\Gamma_2,\emptyset] & (5)\\
a^-_⅋[\Gamma_1]\,a^+_{\mathbf{c}}[\Gamma_2] & \to & T_{\mathbf{c}}[\Gamma_2,\emptyset] & (6)\\
a^+_⅋[\Gamma_1]\,S_\otimes[\Gamma_2,\sigma_2]\,a^-_{\mathbf{c}}[\Gamma_3] & \to & T_{\mathbf{c}}[\Gamma_3,\sigma_2\cup(\Gamma_1\times\Gamma_2)] & (7)\\
 & & \multicolumn{2}{r}{\textit{provided that } (\sigma_2\cup(\Gamma_1\times\Gamma_2))^+ \textit{ is acyclic.}}\\
a^-_⅋[\Gamma_1]\,S_\otimes[\Gamma_2,\sigma_2]\,a^+_{\mathbf{c}}[\Gamma_3] & \to & T_{\mathbf{c}}[\Gamma_3,\sigma_2\cup(\Gamma_1\times\Gamma_2)] & (8)\\
 & & \multicolumn{2}{r}{\textit{provided that } (\sigma_2\cup(\Gamma_1\times\Gamma_2))^+ \textit{ is acyclic.}}\\
a^+_\otimes[\Gamma_1]\,S_⅋[\Gamma_2,\sigma_2]\,a^-_{\mathbf{c}}[\Gamma_3] & \to & T_{\mathbf{c}}[\Gamma_3,\sigma_2\cup(\Gamma_2\times\Gamma_1)] & (9)\\
 & & \multicolumn{2}{r}{\textit{provided that } (\sigma_2\cup(\Gamma_2\times\Gamma_1))^+ \textit{ is acyclic.}}\\
a^-_\otimes[\Gamma_1]\,S_⅋[\Gamma_2,\sigma_2]\,a^+_{\mathbf{c}}[\Gamma_3] & \to & T_{\mathbf{c}}[\Gamma_3,\sigma_2\cup(\Gamma_2\times\Gamma_1)] & (10)\\
 & & \multicolumn{2}{r}{\textit{provided that } (\sigma_2\cup(\Gamma_2\times\Gamma_1))^+ \textit{ is acyclic.}}
\end{array}$$

Proposition 4. *Let $\Gamma \vdash A$ be a Lambek sequent. Then $\Gamma \vdash A$ is provable if and only if*

$$\mathcal{V}_3[\![\Gamma \vdash A]\!] \;\to^*\; S_⅋[\emptyset,\sigma] \qquad (***)$$

according to system R_3, for some $\sigma \subset \mathcal{K}\times\mathcal{K}$ whose transitive closure is acyclic.

Proof. (sketch). By Proposition 3, we have that $\Gamma \vdash A$ is provable if and only if there exists a proof-net of it. Given such a proof-net, it easy to construct a rewriting such as $(***)$ by taking σ to be the dominance relation of the proof-net. This proves the only-if part of the proposition.

Conversely, we show that the set of axiom links induced by $(***)$ satisfies Conditions A, B, and C. To show that Conditions A and B are satisfied, one proceeds similarly to cases of Systems R_1 and R_2. To show that Condition C is satisfied, one establishes the following invariants:

- *the set Γ appearing in a non-terminal symbol $S_⅋[\Gamma,\sigma]$ or $T_⅋[\Gamma,\sigma]$ corresponds the $\otimes$-nodes that are ancestors of the $⅋$-nodes contained in the open face associated to this non-terminal symbol;*
- *the set Γ appearing in a non-terminal symbol $S_\otimes[\Gamma,\sigma]$ or $T_\otimes[\Gamma,\sigma]$ corresponds the $\otimes$-nodes that are contained in the open face associated to this non-terminal symbol;*

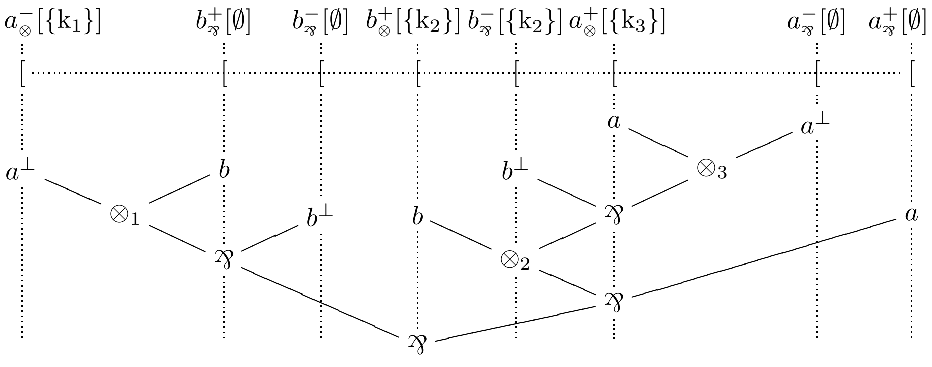
– *the set σ appearing in a non-terminal symbol $S_{\mathbf{c}}[\Gamma, \sigma]$ or $T_{\mathbf{c}}[\Gamma, \sigma]$ corresponds the dominance relation of the module associated to this non-terminal symbol.*

$\square$

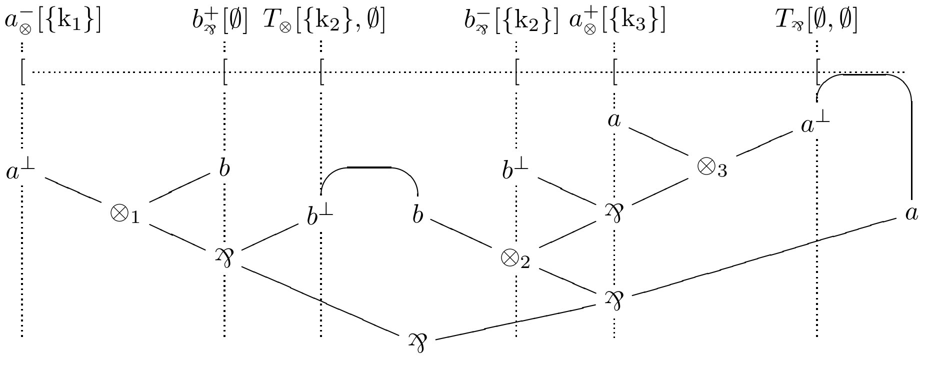
Example 6. Consider again the sequent of Example 1. Its provability may be established by the following rewriting (where applications of Rule (1) have been left implicit):

$$a^-_\otimes[\{k_1\}]\ b^+_{\parr}[\emptyset]\ \underbrace{b^-_{\parr}[\emptyset]\ b^+_\otimes[\{k_2\}]}\ b^-_{\parr}[\{k_2\}]\ a^+_\otimes[\{k_3\}]\ \underbrace{a^-_{\parr}[\emptyset]\ a^+_{\parr}[\emptyset]}$$
$$a^-_\otimes[\{k_1\}]\ \underbrace{b^+_{\parr}[\emptyset]\ T_\otimes[\{k_2\},\emptyset]\ b^-_{\parr}[\{k_2\}]}\ a^+_\otimes[\{k_3\}]\ T_{\parr}[\emptyset,\emptyset]$$
$$\underbrace{a^-_\otimes[\{k_1\}]\ T_{\parr}[\{k_2\},\emptyset]\ a^+_\otimes[\{k_3\}]}\ T_{\parr}[\emptyset,\emptyset]$$
$$\underbrace{T_\otimes[\{k_3\},\{(k_2,k_1)\}]\ T_{\parr}[\emptyset,\emptyset]}$$
$$S_{\parr}[\emptyset,\{(k_2,k_1)\}]$$

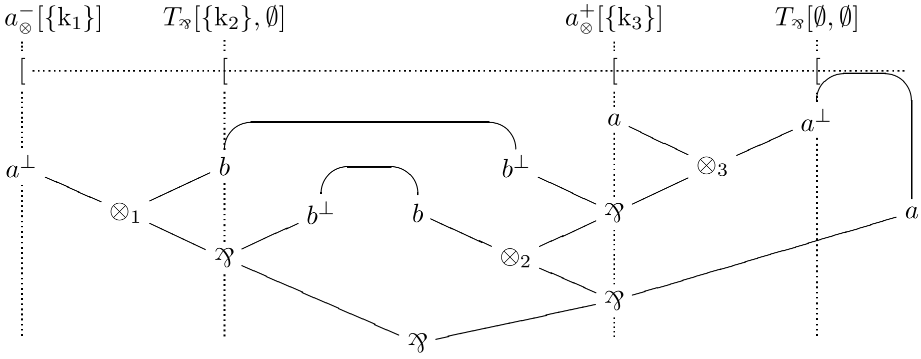
Now, the different rewriting steps of the above derivation may be interpreted as incremental steps in the construction of a proof-net. We start with the proof-frame:



The first rewriting steps, $b^-_{\parr}[\emptyset]\, b^+_\otimes[\{k_2\}] \to T_\otimes[\{k_2\},\emptyset]$ and $a^-_{\parr}[\emptyset]\, a^+_{\parr}[\emptyset] \to T_{\parr}[\emptyset,\emptyset]$, yield the following module:

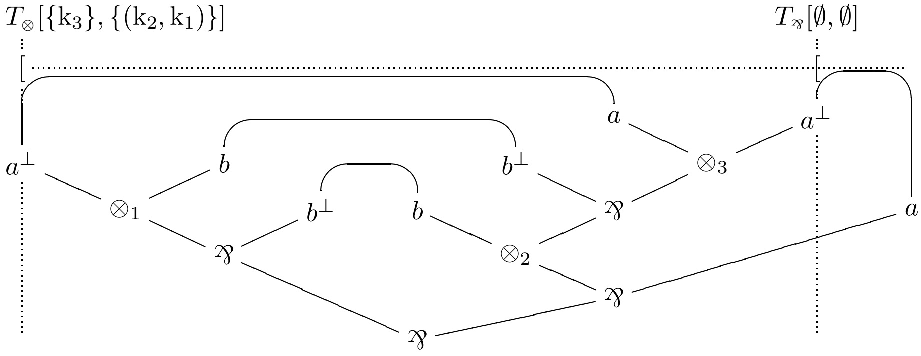


Then, the interpretation of $b^+_{\wp}[\emptyset]\, S_{\otimes}[\{\mathrm{k}_2\},\emptyset]\, b^-_{\wp}[\{\mathrm{k}_2\}] \rightarrow T_{\wp}[\{\mathrm{k}_2\},\emptyset]$ gives rise to a supplementary axiom link:



$a^-_{\otimes}[\{\mathrm{k}_1\}]$
$T_{\wp}[\{\mathrm{k}_2\},\emptyset]$
$a^+_{\otimes}[\{\mathrm{k}_3\}]$
$T_{\wp}[\emptyset,\emptyset]$
a
$a^\perp$
$a^\perp$
b
$b^\perp$
$\otimes_3$
$\otimes_1$
$b^\perp$
b
$\wp$
a
$\wp$
$\otimes_2$
$\wp$
$\wp$

The next step, $a^-_{\otimes}[\{\mathrm{k}_1\}]\, S_{\wp}[\{\mathrm{k}_2\},\emptyset]\, a^+_{\otimes}[\{\mathrm{k}_3\} \rightarrow T_{\otimes}[\{\mathrm{k}_3\},\{(\mathrm{k}_2,\mathrm{k}_1)\}]$, is interpreted similarly:



Finally, the last step, $T_{\otimes}[\{\mathrm{k}_3\},\{(\mathrm{k}_2,\mathrm{k}_1)\}]\, S_{\wp}[\emptyset,\emptyset] \rightarrow S_{\wp}[\emptyset,\{(\mathrm{k}_2,\mathrm{k}_1)\}]$, establishes that the above module is indeed a proof-net.

5 The proof-net construction algorithm

The fact that we have reduced the decision problem of the Lambek calculus to a context-free parsing problem allows us to take advantage of the dynamic programming techniques that are used to recognize the context-free languages. In particular, we present an algorithm derived from the well-known Cocke-Kasami-Younger procedure.

This algorithm, which is based on the Chomsky normal form of System R_3 (See Appendix A), constructs an upper-triangular recognition matrix A whose entries are sets of non-terminal symbols. The initial conditions are the following: Given a sequent $\Gamma \vdash A$, $\omega = \omega_1\omega_2\ldots\omega_n$ contains $\mathcal{V}_3[\![\Gamma \vdash A]\!]$, and all the entries of A are empty.

```
 for $i := 1$ to $n-1$ do
(* Rules 4 and 5 *)
   if      $\omega_i = a^+_{\parr}[\Gamma_1]$ and $\omega_{i+1} = a^-_{\mathbf{c}}[\Gamma_2]$ then $A_{i,i} := \{T_{\mathbf{c}}[\Gamma_2, \emptyset], S_{\mathbf{c}}[\Gamma_2, \emptyset]\}$
(* Rules 6 and 7 *)
   else if $\omega_i = a^-_{\parr}[\Gamma_1]$ and $\omega_{i+1} = a^+_{\mathbf{c}}[\Gamma_2]$ then $A_{i,i} := \{T_{\mathbf{c}}[\Gamma_2, \emptyset], S_{\mathbf{c}}[\Gamma_2, \emptyset]\}$
 od;
 for $d := 2$ to $n-1$ do
   if $d$ is even
   then for $i := 1$ to $n-d$ do
          $j := (d+i)-1$;
(* Rule 8 *)
          if      $\omega_i = a^+_{\parr}[\Gamma_1]$ then for each $S_{\otimes}[\Gamma_2, \sigma_2] \in A_{i+1,j}$ do
                                              $\sigma := \sigma_2 \cup (\Gamma_1 \times \Gamma_2)$;
                                              if $\sigma^+$ is acyclic
                                                then $A_{i,j} := A_{i,j} \cup \{U[a, +, \sigma]\}$
                                            od
(* Rule 9 *)
          else if $\omega_i = a^-_{\parr}[\Gamma_1]$ then for each $S_{\otimes}[\Gamma_2, \sigma_2] \in A_{i+1,j}$ do
                                              $\sigma := \sigma_2 \cup (\Gamma_1 \times \Gamma_2)$;
                                              if $\sigma^+$ is acyclic
                                                then $A_{i,j} := A_{i,j} \cup \{U[a, -, \sigma]\}$
                                            od
(* Rule 10 *)
          else if $\omega_i = a^+_{\otimes}[\Gamma_1]$ then for each $S_{\parr}[\Gamma_2, \sigma_2] \in A_{i+1,j}$ do
                                              $\sigma := \sigma_2 \cup (\Gamma_2 \times \Gamma_1)$;
                                              if $\sigma^+$ is acyclic
                                                then $A_{i,j} := A_{i,j} \cup \{U[a, +, \sigma]\}$
                                            od
(* Rule 11 *)
          else if $\omega_i = a^-_{\otimes}[\Gamma_1]$ then for each $S_{\parr}[\Gamma_2, \sigma_2] \in A_{i+1,j}$ do
                                              $\sigma := \sigma_2 \cup (\Gamma_2 \times \Gamma_1)$;
                                              if $\sigma^+$ is acyclic
                                                then $A_{i,j} := A_{i,j} \cup \{U[a, -, \sigma]\}$
                                            od
        od
   else (* $d$ is odd *)
        for $i := 1$ to $n-d$ do
          $j := (d+i)-1$;
(* Rules 12 and 13 *)
          for each $U[a, +, \sigma] \in A_{i,j-1}$ do
            if $\omega_{j+1} = a^-_{\mathbf{c}}[\Gamma]$ then $A_{i,j} := A_{i,j} \cup \{T_{\mathbf{c}}[\Gamma, \sigma], S_{\mathbf{c}}[\Gamma, \sigma]\}$
          od;
(* Rules 14 and 15 *)
          for each $U[a, -, \sigma] \in A_{i,j-1}$ do
            if $\omega_{j+1} = a^+_{\mathbf{c}}[\Gamma]$ then $A_{i,j} := A_{i,j} \cup \{T_{\mathbf{c}}[\Gamma, \sigma], S_{\mathbf{c}}[\Gamma, \sigma]\}$
```

```
                od;
                $k := i$;
                while $k < j$ do
(* Rule 2 *)
                  for each $T_{⅋}[\Gamma_1, \sigma_1] \in A_{i,k}$ do
                    for each $S_{\otimes}[\Gamma_2, \sigma_2] \in A_{k+2,j}$ do
                      $\sigma := \sigma_1 \cup \sigma_2 \cup (\Gamma_1 \times \Gamma_2)$;
                      if $\sigma^+$ is acyclic then $A_{i,j} := A_{i,j} \cup \{S_{⅋}[\Gamma_1, \sigma]\}$
                    od
                  od;
(* Rules 1 and 3 *)
                  for each $T_{\otimes}[\Gamma_1, \sigma_1] \in A_{i,k}$ do
                    for each $S_{⅋}[\Gamma_2, \sigma_2] \in A_{k+2,j}$ do
                      $\sigma := \sigma_1 \cup \sigma_2 \cup (\Gamma_2 \times \Gamma_1)$;
                      if $\sigma^+$ is acyclic then $A_{i,j} := A_{i,j} \cup \{S_{⅋}[\Gamma_2, \sigma]\}$
                    od
                    for each $S_{\otimes}[\Gamma_2, \sigma_2] \in A_{k+2,j}$ do
                      $\sigma := \sigma_1 \cup \sigma_2$;
                      if $\sigma^+$ is acyclic then $A_{i,j} := A_{i,j} \cup \{S_{\otimes}[\Gamma_1 \cup \Gamma_2, \sigma]\}$
                    od
                  od;
                  $k := k + 2$
                od
              od
  od
```

6 Conclusions and future work

Our proof-net construction algorithm does not take any advantage of the intuitionistic nature of the Lambek calculus. Consequently it is easily adaptable to classical calculus such as Yetter's [11] and Abrusci's [1]

Our work indirectly addresses the problem of the complexity of the Lambek calculus decision problem. Whether this problem is NP-hard or not is still open. The theoretical complexity of our algorithm is exponential because of the number of possible non-terminal symbols. It is possible to reduce this number because it is not necessary to record the complete dominance relation but only the constraints that are "still active". However, this optimisation, which we will describe in an extended version of this paper, does not give rise to a polynomial algorithm. Nonetheless, it allows to define interesting fragments of the Lambek calculus for which the decision problem is polynomial.

Acknowledgements I wish to thank Glyn Morrill, François Lamarche, and Yves Lafont for helpful discussions and comments.

References

1. M. Abrusci. Phase semantics and sequent calculus for pure non-commutative classical linear logic. *Journal of Symbolic Logic*, 56(4):1403–1451, 1991.
2. C. Berge. *Graphs.* North-Holland, second revised edition edition, 1985.
3. A Fleury. *La règle d'échange : logique lineaire multiplicative tressée.* Thès de Doctorat, spécialité Mathématiques, Université Paris 7, 1996.
4. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
5. J.-Y. Girard. Quantifiers in linear logic II. Technical Report 19, Equipe de Logique Mathématique, Université de Paris VII, 1991.
6. F. Lamarche and C. Retoré. Proof nets for the Lambek calculus. In M. Abrusci and C. Casadio, editors, *Proofs and Linguistic Categories, Proceedings 1996 Roma Workshop.* Cooperativa Libraria Universitaria Editrice Bologna, 1996.
7. J. Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–170, 1958.
8. M. Moortgat. Categorial type logic. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2. Elsevier, 1997.
9. G. Morrill. Memoisation of categorial proof nets: parallelism in categorial processing. In V. M. Abrusci and C. Casadio, editors, *Proofs and Linguistic Categories, Proceedings 1996 Roma Workshop.* Cooperativa Libraria Universitaria Editrice Bologna, 1996.
10. D. Roorda. *Resource Logics: proof-theoretical investigations.* PhD thesis, University of Amsterdam, 1991.
11. D. N. Yetter. Quantales and (non-commutative) linear logic. *Journal of Symbolic Logic*, 55:41–64, 1990.

A Chomsky normal form of System $\boldsymbol{R}_3$

$$T_\otimes[\Gamma_1,\sigma_1]\, S_\parr[\Gamma_2,\sigma_2] \;\rightarrow\; S_\parr[\Gamma_2,\sigma_1\cup\sigma_2\cup(\Gamma_2\times\Gamma_1)] \tag{1}$$

$$T_\parr[\Gamma_1,\sigma_1]\, S_\otimes[\Gamma_2,\sigma_2] \;\rightarrow\; S_\parr[\Gamma_1,\sigma_1\cup\sigma_2\cup(\Gamma_1\times\Gamma_2)] \tag{2}$$

$$T_\otimes[\Gamma_1,\sigma_1]\, S_\otimes[\Gamma_2,\sigma_2] \;\rightarrow\; S_\otimes[\Gamma_1\cup\Gamma_2,\sigma_1\cup\sigma_2] \tag{3}$$

$$a^+_\parr[\Gamma_1]\, a^-_{\mathbf{c}}[\Gamma_2] \;\rightarrow\; T_{\mathbf{c}}[\Gamma_2,\emptyset] \tag{4}$$

$$a^+_\parr[\Gamma_1]\, a^-_{\mathbf{c}}[\Gamma_2] \;\rightarrow\; S_{\mathbf{c}}[\Gamma_2,\emptyset] \tag{5}$$

$$a^-_\parr[\Gamma_1]\, a^+_{\mathbf{c}}[\Gamma_2] \;\rightarrow\; T_{\mathbf{c}}[\Gamma_2,\emptyset] \tag{6}$$

$$a^-_\parr[\Gamma_1]\, a^+_{\mathbf{c}}[\Gamma_2] \;\rightarrow\; S_{\mathbf{c}}[\Gamma_2,\emptyset] \tag{7}$$

$$a^+_\parr[\Gamma_1]\, S_\otimes[\Gamma_2,\sigma_2] \;\rightarrow\; U[a,+,\sigma_2\cup(\Gamma_1\times\Gamma_2)] \tag{8}$$

$$a^-_\parr[\Gamma_1]\, S_\otimes[\Gamma_2,\sigma_2] \;\rightarrow\; U[a,-,\sigma_2\cup(\Gamma_1\times\Gamma_2)] \tag{9}$$

$$a^+_\otimes[\Gamma_1]\, S_\parr[\Gamma_2,\sigma_2] \;\rightarrow\; U[a,+,\sigma_2\cup(\Gamma_2\times\Gamma_1)] \tag{10}$$

$$a^-_\otimes[\Gamma_1]\, S_\parr[\Gamma_2,\sigma_2] \;\rightarrow\; U[a,-,\sigma_2\cup(\Gamma_2\times\Gamma_1)] \tag{11}$$

$$U[a,+,\sigma]\, a^-_{\mathbf{c}}[\Gamma] \;\rightarrow\; T_{\mathbf{c}}[\Gamma,\sigma] \tag{12}$$

$$U[a,+,\sigma]\, a^-_{\mathbf{c}}[\Gamma] \;\rightarrow\; S_{\mathbf{c}}[\Gamma,\sigma] \tag{13}$$

$$U[a,-,\sigma]\, a^+_{\mathbf{c}}[\Gamma] \;\rightarrow\; T_{\mathbf{c}}[\Gamma,\sigma] \tag{14}$$

$$U[a,-,\sigma]\, a^+_{\mathbf{c}}[\Gamma] \;\rightarrow\; S_{\mathbf{c}}[\Gamma,\sigma] \tag{15}$$

Tractable Transformations from Modal Provability Logics into First-Order Logic

Stéphane Demri[1]* and Rajeev Goré[2]**

[1] Laboratoire LEIBNIZ - C.N.R.S.
46 av. Félix Viallet, 38000 Grenoble, France
demri@imag.fr

[2] Automated Reasoning Project and Dept. of Computer Science
Australian National University, Canberra 0200, Australia
rpg@arp.anu.edu.au

Abstract. We define a class of modal logics LF by uniformly extending a class of modal logics L. Each logic L is characterised by a class of first-order definable frames, but the corresponding logic LF is sometimes characterised by classes of modal frames that are not first-order definable. The class LF includes provability logics with deep arithmetical interpretations. Using Belnap's proof-theoretical framework Display Logic we characterise the "pseudo-displayable" subclass of LF and show how to define polynomial-time transformations from each such LF into the corresponding L, and hence into first-order classical logic. Theorem provers for classical first-order logic can then be used to mechanise deduction in these "psuedo-displayable second order" modal logics.

1 Introduction

Background. There are two main approaches to modal theorem proving in the literature. The *direct approach* consists in defining calculi dedicated to modal logics at the cost of modifying existing theorem provers (see e.g. [Fit83,AEH90,Mas94]). The *translational approach* consists in translating modal logics into logics for which theorem provers already exist, typically classical first-order logic (FOL). The relational translation into FOL (see e.g. [Mor76,Sch99,GHM98]) is the most common such translation although not the only one (see e.g. [Mor76,Ohl88,Her89,dMP95,Ohl98]). These two approaches cannot always be applied with equal success (see e.g. [HS97]). For instance, for the provability logic G which is characterized by a second-order class of modal frames (see e.g. [Boo93]), the relational translation is not possible unless FOL is augmented with fixed-point operators (see e.g. [NS98]). However, dedicated sequent-style calculi do exist for provability logics such as G or Grz (for Grzegorczyk); see e.g. [SV80,Lei81,Fit83,Val83,Avr84,Boo93,Gor99].

Display Logic. Display Logic (**DL**) [Bel82] is a proof-theoretical framework that generalises the structural language of Gentzen's sequents by using multiple

* Visit to ARP supported by an Australian Research Council International Fellowship.
** Supported by an Australian Research Council Queen Elizabeth II Fellowship.

H. Ganzinger (Ed.): CADE-16, LNAI 1632, pp. 16–30, 1999.

complex structural connectives instead of Gentzen's comma. The term "display" comes from the nice property that any occurrence of a *structure* in a sequent can be displayed either as the entire antecedent or as the entire succedent of some sequent which is *structurally equivalent* to the initial sequent (see e.g. [Bel82]). Any display calculus satisfying the conditions (C1)-(C8) [Bel82] (see the appendix) enjoys cut-elimination. Kracht's characterisation of *properly displayable* modal logics [Kra96] means that any extension of the (poly)modal logic K obtained by the addition of the so-called *primitive axioms* admits a display calculus that obeys conditions (C1)-(C8) [Kra96], and therefore enjoys cut-elimination. Since every primitive axiom is a Sahlqvist formula, any such extension is first-order definable [Sah75].

Our Contribution. Let ϕ be a modal formula and $\mathtt{F}(\phi)$ be a formula built from $\{\phi\}$ using $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Box$ and $\Diamond$ such that any subformula of the form $\Box\psi$ in $\mathtt{F}(\phi)$ occurs positively (when every $\phi_1 \Rightarrow \phi_2$ is written as $\neg\phi_1 \vee \phi_2$). Let $\mathtt{L}$ be a properly displayable modal logic and $\mathtt{LF}$ be the logic obtained from $\mathtt{L}$ by adding the axiom scheme $\Box(\mathtt{F}(\phi) \Rightarrow \phi) \Rightarrow \Box\phi$. Here a *logic* is understood as a set of formulae and therefore is exactly a *(decision) problem* in the usual sense in complexity theory. That is, as a language viewed as a set of strings built upon a given alphabet.

By generalising results from [DG99a], we establish conditions permitting an $\mathcal{O}(n^3.log\ n)$-time transformation (also called a "many-one reduction" [Pap94]) g from $\mathtt{LF}$ into $\mathtt{L}$. If K4 $\subseteq \mathtt{L}$, then g can be in $\mathcal{O}(n.log\ n)$-time. Now, every primitive modal logic can be translated into FOL in linear-time (using a smart recycling of the variables). So in the general case, we define an $\mathcal{O}(n^3.log\ n)$-time transformation from $\mathtt{LF}$ into (possibly known) fragments of FOL even though a formula of second-order logic may be essential to describe the class of modal frames characterising $\mathtt{LF}$. This provides an alternative for mechanizing modal provability logics.

Our uniform definition of such mappings shows that **DL** is ideal for proof-theoretical analyses of calculi for $\mathtt{LF}$ and $\mathtt{L}$. In fact, the theoremhood preserving nature of our transformations are a characterisation of (weak) cut-elimination for many of these logics.

Plan of the Paper. In Section 2, we define the class of modal logics $\mathtt{LF}$ studied in the paper. In Section 3, we define display calculi $\delta\mathtt{LF}$ for these logics and show these calculi to be sound and complete. In Section 4, we give necessary and sufficient conditions to establish that the display calculi $\delta\mathtt{LF}$ admit a (weak) cut-elimination theorem, and provide the promised transformations. Section 5 contains a similar analysis for traditional sequent-style calculi. Proofs are omitted because of lack of space and they can be found in [DG99b].

2 Provability Logics

Given a set $\mathtt{PRP} = \{\mathtt{p}_1, \mathtt{p}_2, \ldots\}$ of *atomic formulae*, the formulae $\phi \in \mathtt{FML}$ are inductively defined as follows for $\mathtt{p}_i \in \mathtt{PRP}$:

$$\phi \ ::= \bot \mid \top \mid \mathtt{p}_i \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \phi_1 \Rightarrow \phi_2 \mid \Box\phi$$

Standard abbreviations include $\Leftrightarrow$, $\Diamond$: for instance, $\Diamond\phi \overset{\text{def}}{=} \neg\Box\neg\phi$. An occurrence of the subformula ψ in ϕ is *positive* [resp. *negative*] iff it is in the scope of an even [resp. odd] number of negations, where as usual, every occurrence of $\phi_1 \Rightarrow \phi_2$ is treated as an occurence of $\neg\phi_1 \vee \phi_2$. The standard axiomatic Hilbert system K consists of the axiom schemes

1. the tautologies of the Propositional Calculus (PC)
2. $\Box(\mathtt{p} \Rightarrow \mathtt{q}) \Rightarrow (\Box\mathtt{p} \Rightarrow \Box\mathtt{q})$

and the inference rules: *modus ponens* (from ϕ and $\phi \Rightarrow \psi$ infer ψ) and *necessitation* (from ϕ infer $\Box\phi$). When L is an extension of K (including K), we write $\phi \in \mathtt{L}$ to denote that ϕ is a theorem of L. In the paper, we refer to the following well-known extensions L of K:

$\mathrm{T} \overset{\text{def}}{=} \mathrm{K} + \Box\mathtt{p} \Rightarrow \mathtt{p}$ $\qquad$ $\mathrm{K4} \overset{\text{def}}{=} \mathrm{K} + \Box\mathtt{p} \Rightarrow \Box\Box\mathtt{p}$
$\mathrm{S4} \overset{\text{def}}{=} \mathrm{K4} + \Box\mathtt{p} \Rightarrow \mathtt{p}$ $\qquad$ $\mathrm{G} \overset{\text{def}}{=} \mathrm{K4} + \Box(\Box\mathtt{p} \Rightarrow \mathtt{p}) \Rightarrow \Box\mathtt{p}$
$\mathrm{Grz} \overset{\text{def}}{=} \mathrm{S4} + \Box(\Box(\mathtt{p} \Rightarrow \Box\mathtt{p}) \Rightarrow \mathtt{p}) \Rightarrow \Box\mathtt{p}$

Following [Kra96], a formula is *primitive* iff it is of the form $\phi \Rightarrow \psi$ where both ϕ and ψ are built from $\mathtt{PRP} \cup \{\top\}$ with the help of $\wedge$, $\vee$, $\Diamond$, and where ϕ contains each atomic proposition at most once.

Example 1. Neither of the formulae $\Box\mathtt{p} \Rightarrow \mathtt{p}$ and $\Box\mathtt{p} \Rightarrow \Box\Box\mathtt{p}$ are primitive, but their logically equivalent (in K) forms $\mathtt{p} \Rightarrow \Diamond\mathtt{p}$ and $\Diamond\Diamond\mathtt{p} \Rightarrow \Diamond\mathtt{p}$ are both primitive.

Definition 1. *[Kra96] A logic (defined via Hilbert system)* L *is* properly displayable *iff* L *is obtained from K by adding primitive formulae as axioms.*

By Example 1, the logics T, K4 and S4 are properly displayable. In general, many of the traditional axioms for the most well-known modal logics are not primitive, but most of them have a primitive equivalent [Kra96]. In [Kra96], it is shown that every properly displayable logic admits a display calculus satisfying the conditions (C1)-(C8) [Bel82] and therefore enjoys cut-elimination. In what follows, we write $\delta\mathtt{L}$ to denote the display calculus for L defined in [Kra96].

A *formula generation map* $\mathtt{F} : \mathtt{FML} \rightarrow \mathtt{FML}$ is an application such that there is a formula $\psi_{\mathtt{F}}$ containing only one atomic proposition, say p, and no logical constants, such that for $\phi \in \mathtt{FML}$, $\mathtt{F}(\phi)$ is obtained from $\psi_{\mathtt{F}}$ by replacing every occurrence of p by ϕ. Moreover, we assume that no subformula of the form $\Box\varphi$ occurs negatively in $\psi_{\mathtt{F}}$. F is also written $\lambda\mathtt{p}.\psi_{\mathtt{F}}$. For any properly displayable logic L and any formula generation map F, we write LF to denote the logic obtained from L by addition of the scheme

$$\Box(\mathtt{F}(\mathtt{p}) \Rightarrow \mathtt{p}) \Rightarrow \Box\mathtt{p} \tag{1}$$

Observe that $\Box(\mathtt{F}(\mathtt{q}) \Rightarrow \mathtt{q}) \Rightarrow \Box\mathtt{q}$ is not a Sahlqvist formula. This does not exclude the possibility to find a Sahlqvist formula logically equivalent (in the basic modal logic K) to it. For instance, this is the case when $\mathtt{F}(\mathtt{q}) = \neg\mathtt{q}$ since then, $\Box(\mathtt{F}(\mathtt{q}) \Rightarrow \mathtt{q})) \Rightarrow \Box\mathtt{q}$ is just $\Box\mathtt{q} \Rightarrow \Box\mathtt{q}$, and this has an equivalent primitive form $\top \Rightarrow \top$. Moreover, in numerous cases $\mathtt{LF}$ is not properly displayable. For instance, let $\mathtt{F}_G$ and $\mathtt{F}_{Grz}$ be $\lambda\mathtt{p}.\Box\mathtt{p}$ and $\lambda\mathtt{p}.\Box(\mathtt{p} \Rightarrow \Box\mathtt{p})$ respectively. Then, by definition $\mathrm{G} = \mathrm{K4}\mathtt{F}_G$ and $\mathrm{Grz} = \mathrm{S4}\mathtt{F}_{Grz}$. Since $\mathtt{F}_G$ and $\mathtt{F}_{Grz}$ are modal axioms that correspond to essentially second-order conditions on frames, the logics G and Grz are not properly displayable in the sense of Definition 1.

3 Display Calculi

In this section, $\mathtt{L}$ is a properly displayable logic, $\mathtt{F}$ is a formula generation map and $\mathtt{LF}$ is the corresponding extension of $\mathtt{L}$ by the axiom scheme (1).

We briefly recall the main features of the calculus $\delta\mathtt{L}$ from [Wan94,Kra96]. On the structural side, we have structural connectives $*$ (unary), $\circ$ (binary), I (nullary) and $\bullet$ (unary). A *structure* $\mathtt{X} \in \mathtt{struc}(\delta\mathtt{L})$ is inductively defined as follows for $\phi \in \mathtt{FML}$:

$$\mathtt{X} ::= \phi \mid *\mathtt{X} \mid \mathtt{X}_1 \circ \mathtt{X}_2 \mid I \mid \bullet\mathtt{X}$$

We assume that the unary connectives bind tighter than the binary ones. We use *formula variables* like ϕ, ψ, φ to stand for formulae, and use *structure variables* like $\mathtt{X}$, $\mathtt{Y}$ and $\mathtt{Z}$ to stand for arbitrary structures from $\mathtt{struc}(\delta\mathtt{L})$.

A *sequent* is a pair of structures of the form $\mathtt{X} \vdash \mathtt{Y}$ with $\mathtt{X}$ the *antecedent* and $\mathtt{Y}$ the *succedent*. The rules of $\delta\mathtt{L}$ are presented in Figures 1-4. Additional structural rules satisfying conditions (C1)-(C8) are also needed but are omitted here since they depend on the primitive axioms defining $\mathtt{L}$ (see [Kra96] for details).

$$\text{(Id)}\ \ \mathtt{p} \vdash \mathtt{p} \qquad\qquad \text{(cut)}\ \ \frac{\mathtt{X} \vdash \phi \qquad \phi \vdash \mathtt{Y}}{\mathtt{X} \vdash \mathtt{Y}}$$

Fig. 1. Fundamental logical axioms and cut rule

The *display postulates* (reversible rules) in Figure 2 deal with the manipulation of structural connectives.

In any structure $\mathtt{Z}$, the structure $\mathtt{X}$ occurs *negatively* [resp. *positively*] iff $\mathtt{X}$ occurs in the scope of an odd number [resp. an even number] of occurrences of $*$ [Bel82]. In a sequent $\mathtt{Z} \vdash \mathtt{Z}'$, an occurrence of $\mathtt{X}$ is an *antecedent part* [resp. *succedent part*] iff it occurs positively in $\mathtt{Z}$ [resp. negatively in $\mathtt{Z}'$] or it occurs negatively in $\mathtt{Z}'$ [resp. positively in $\mathtt{Z}$] [Bel82]. Two sequents $\mathtt{X} \vdash \mathtt{Y}$ and $\mathtt{X}' \vdash \mathtt{Y}'$ are *structurally equivalent* iff there is a derivation of the first sequent from the second (and vice-versa) only using display postulates from Figure 2.

Theorem 1. *([Bel82]) For every sequent* $\mathtt{X} \vdash \mathtt{Y}$ *and every antecedent [resp. succedent] part* $\mathtt{Z}$ *of* $\mathtt{X} \vdash \mathtt{Y}$*, there is a structurally equivalent sequent* $\mathtt{Z} \vdash \mathtt{Y}'$ *[resp.* $\mathtt{X}' \vdash \mathtt{Z}$*] that has* $\mathtt{Z}$ *(alone) as its antecedent [resp. succedent].* $\mathtt{Z}$ *is said to be* displayed *in* $\mathtt{Z} \vdash \mathtt{Y}'$ *[resp.* $\mathtt{X}' \vdash \mathtt{Z}$*].*

$$\frac{\mathtt{X} \circ \mathtt{Y} \vdash \mathtt{Z}}{\mathtt{X} \vdash \mathtt{Z} \circ *\mathtt{Y}} \qquad \frac{\mathtt{X} \circ \mathtt{Y} \vdash \mathtt{Z}}{\mathtt{Y} \vdash *\mathtt{X} \circ \mathtt{Z}} \qquad \frac{\mathtt{X} \vdash \mathtt{Y} \circ \mathtt{Z}}{\mathtt{X} \circ *\mathtt{Z} \vdash \mathtt{Y}} \qquad \frac{\mathtt{X} \vdash \mathtt{Y} \circ \mathtt{Z}}{*\mathtt{Y} \circ \mathtt{X} \vdash \mathtt{Z}}$$

$$\frac{*\mathtt{X} \vdash \mathtt{Y}}{*\mathtt{Y} \vdash \mathtt{X}} \qquad \frac{\mathtt{X} \vdash *\mathtt{Y}}{\mathtt{Y} \vdash *\mathtt{X}} \qquad \frac{**\mathtt{X} \vdash \mathtt{Y}}{\mathtt{X} \vdash \mathtt{Y}} \qquad \frac{\mathtt{X} \vdash **\mathtt{Y}}{\mathtt{X} \vdash \mathtt{Y}} \qquad \frac{\mathtt{X} \vdash \bullet\mathtt{Y}}{\bullet\mathtt{X} \vdash \mathtt{Y}}$$

Fig. 2. Display postulates

$$\frac{}{\bot \vdash I}\ (\bot\vdash) \qquad \frac{\mathtt{X} \vdash I}{\mathtt{X} \vdash \bot}\ (\vdash\bot) \qquad \frac{I \vdash \mathtt{X}}{\top \vdash \mathtt{X}}\ (\top\vdash) \qquad \frac{}{I \vdash \top}\ (\vdash\top)$$

$$\frac{\mathtt{X} \vdash \phi \quad \psi \vdash \mathtt{Y}}{\phi \Rightarrow \psi \vdash *\mathtt{X} \circ \mathtt{Y}}\ (\Rightarrow\vdash) \qquad \frac{\mathtt{X} \circ \phi \vdash \psi}{\mathtt{X} \vdash \phi \Rightarrow \psi}\ (\vdash\Rightarrow)$$

$$\frac{\phi \circ \psi \vdash \mathtt{X}}{\phi \wedge \psi \vdash \mathtt{X}}\ (\wedge\vdash) \qquad \frac{\mathtt{X} \vdash \phi \quad \mathtt{Y} \vdash \psi}{\mathtt{X} \circ \mathtt{Y} \vdash \phi \wedge \psi}\ (\vdash\wedge) \qquad \frac{*\phi \vdash \mathtt{X}}{\neg\phi \vdash \mathtt{X}}\ (\neg\vdash) \qquad \frac{\mathtt{X} \vdash *\phi}{\mathtt{X} \vdash \neg\phi}\ (\vdash\neg)$$

$$\frac{\mathtt{X} \vdash \bullet\phi}{\mathtt{X} \vdash \Box\phi}\ (\vdash\Box_{\mathtt{L}}) \qquad \frac{\phi \vdash \mathtt{X}}{\Box\phi \vdash \bullet\mathtt{X}}\ (\Box\vdash) \qquad \frac{\phi \vdash \mathtt{X} \quad \psi \vdash \mathtt{Y}}{\phi \vee \psi \vdash \mathtt{X} \circ \mathtt{Y}}\ (\vee\vdash) \qquad \frac{\mathtt{X} \vdash \phi \circ \psi}{\mathtt{X} \vdash \phi \vee \psi}\ (\vdash\vee)$$

Fig. 3. Operational rules

Theorem 2. *[Kra96] For all $\phi \in \mathtt{FML}$, $I \vdash \phi$ has a cut-free proof in $\delta\mathtt{L}$ iff $\phi \in \mathtt{L}$.*

To define the calculus $\delta\mathtt{LF}$ we need one additional notion. Let m be a map $m : \mathtt{FML} \times \{0,1\} \rightarrow \mathtt{struc}(\delta\mathtt{L})$ that transforms certain logical connectives into structural connectives, inductively defined as follows ($i \in \{0,1\}$):

$$m(\mathtt{p}, i) \stackrel{\text{def}}{=} \mathtt{p} \text{ for any } \mathtt{p} \in \mathtt{PRP}$$

$$\begin{array}{ll}
m(\top, i) \stackrel{\text{def}}{=} \top & m(\bot, i) \stackrel{\text{def}}{=} \bot \\
m(\phi_1 \vee \phi_2, 0) \stackrel{\text{def}}{=} \phi_1 \vee \phi_2 & m(\phi_1 \vee \phi_2, 1) \stackrel{\text{def}}{=} m(\phi_1, 1) \circ m(\phi_2, 1) \\
m(\phi_1 \wedge \phi_2, 0) \stackrel{\text{def}}{=} m(\phi_1, 0) \circ m(\phi_2, 0) & m(\phi_1 \wedge \phi_2, 1) \stackrel{\text{def}}{=} \phi_1 \wedge \phi_2 \\
m(\phi_1 \Rightarrow \phi_2, 0) \stackrel{\text{def}}{=} \phi_1 \Rightarrow \phi_2 & m(\phi_1 \Rightarrow \phi_2, 1) \stackrel{\text{def}}{=} *m(\phi_1, 0) \circ m(\phi_2, 1) \\
m(\Box\phi, i) \stackrel{\text{def}}{=} \Box\phi & m(\neg\phi, i) \stackrel{\text{def}}{=} *m(\phi, 1-i).
\end{array}$$

The second argument of m indicates when the first argument is read as an antecedent part ($i = 0$) or as a succedent part ($i = 1$). The calculus $\delta\mathtt{LF}$ has the same structures as $\delta\mathtt{L}$, and is obtained from $\delta\mathtt{L}$ by replacing the ($\vdash \Box_{\mathtt{L}}$)-rule from Figure 3 by the ($\vdash \Box_{\mathtt{LF}}$) rule:

$$\frac{\mathtt{X} \vdash \bullet(*m(\mathtt{F}(\phi), 0) \circ \phi)}{\mathtt{X} \vdash \Box\phi}\ (\vdash \Box_{\mathtt{LF}})$$

$$\frac{\mathtt{X} \vdash \mathtt{Z}}{I \circ \mathtt{X} \vdash \mathtt{Z}}\,(I_l) \qquad \frac{\mathtt{X} \vdash \mathtt{Z}}{\mathtt{X} \vdash I \circ \mathtt{Z}}\,(I_r) \qquad \frac{I \vdash \mathtt{Y}}{*I \vdash \mathtt{Y}}\,(Q_l) \qquad \frac{\mathtt{X} \vdash I}{\mathtt{X} \vdash *I}\,(Q_r)$$

$$\frac{\mathtt{X} \vdash \mathtt{Z}}{\mathtt{Y} \circ \mathtt{X} \vdash \mathtt{Z}}\,(weak_l) \qquad \frac{\mathtt{X} \vdash \mathtt{Z}}{\mathtt{X} \circ \mathtt{Y} \vdash \mathtt{Z}}\,(weak_r) \qquad \frac{I \vdash \mathtt{X}}{\bullet I \vdash \mathtt{X}}\,(nec^l) \qquad \frac{\mathtt{X} \vdash I}{\mathtt{X} \vdash \bullet I}\,(nec^r)$$

$$\frac{\mathtt{X}_1 \circ (\mathtt{X}_2 \circ \mathtt{X}_3) \vdash \mathtt{Z}}{(\mathtt{X}_1 \circ \mathtt{X}_2) \circ \mathtt{X}_3 \vdash \mathtt{Z}}\,(assoc_l) \qquad \frac{\mathtt{Z} \vdash \mathtt{X}_1 \circ (\mathtt{X}_2 \circ \mathtt{X}_3)}{\mathtt{Z} \vdash (\mathtt{X}_1 \circ \mathtt{X}_2) \circ \mathtt{X}_3}\,(assoc_r)$$

$$\frac{\mathtt{Y} \circ \mathtt{X} \vdash \mathtt{Z}}{\mathtt{X} \circ \mathtt{Y} \vdash \mathtt{Z}}\,(com_l) \qquad \frac{\mathtt{Z} \vdash \mathtt{Y} \circ \mathtt{X}}{\mathtt{Z} \vdash \mathtt{X} \circ \mathtt{Y}}\,(com_r) \qquad \frac{\mathtt{X} \circ \mathtt{X} \vdash \mathtt{Y}}{\mathtt{X} \vdash \mathtt{Y}}\,(contr_l) \qquad \frac{\mathtt{Y} \vdash \mathtt{X} \circ \mathtt{X}}{\mathtt{Y} \vdash \mathtt{X}}\,(contr_r)$$

Fig. 4. Other basic structural rules

The ($\vdash \Box_{\mathtt{LF}}$)-rules for δGrz and δG are respectively:

$$\frac{\mathtt{X} \vdash \bullet(*\Box(\phi \Rightarrow \Box\phi) \circ \phi)}{\mathtt{X} \vdash \Box\phi}\,(\vdash \Box_{Grz}) \qquad \frac{\mathtt{X} \vdash \bullet(*\Box\phi \circ \phi)}{\mathtt{X} \vdash \Box\phi}\,(\vdash \Box_G)$$

The calculus $\delta\mathtt{LF}$ satisfies conditions (C2)-(C7). In particular, δG satisfies the conditions (C1)-(C7). The ($\vdash \Box_G$)-rule in δG is similar to the GLR rule in [SV82] (see also [Avr84]). Analogously, the ($\vdash \Box_{Grz}$)-rule in δGrz is similar to the (GRZc) rule in [BG86] or to the ($\Rightarrow \Box$) rule in [Avr84]. An intuitively obvious way to understand the ($\vdash \Box_{\mathtt{LF}}$)-rule is to recall the *double nature* of the $\Box$-formulae in $\mathtt{LF}$ as illustrated by the $\mathtt{LF}$-theorem $\Box\phi \Leftrightarrow \Box(\mathtt{F}(\phi) \Rightarrow \phi)$.

We use the label (*dp*) as shown below left to denote that the sequent s' is obtained from the sequent s by an unspecified finite number (possibly zero) of display postulate applications from Figure 2. The ($\vdash \Box_{\mathtt{L}}$)-rule from $\delta\mathtt{L}$ is derivable in $\delta\mathtt{LF}$ as shown below right:

$$\frac{s}{s'}\,(dp) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\mathtt{X} \vdash \bullet\phi}{\bullet\mathtt{X} \vdash \phi}\,(dp)}{m(\mathtt{F}(\phi), 0) \circ \bullet\mathtt{X} \vdash \phi}\,(weak_l)}{\mathtt{X} \vdash \bullet(*m(\mathtt{F}(\phi), 0) \circ \phi)}\,(dp)}{\mathtt{X} \vdash \Box\phi}\,(\vdash \Box_{\mathtt{LF}})$$

To prove soundness of $\delta\mathtt{LF}$ with respect to $\mathtt{LF}$-theoremhood, we use the mappings $a : \mathtt{struc}(\delta\mathtt{L}) \to \mathtt{FML}$ and $s : \mathtt{struc}(\delta\mathtt{L}) \to \mathtt{FML}$ recalled below:

$$a(\phi) \stackrel{\text{def}}{=} s(\phi) \stackrel{\text{def}}{=} \phi \text{ for any } \phi \in \mathtt{FML}$$

$$\begin{array}{llll}
a(I) & \stackrel{\text{def}}{=} \top & s(I) & \stackrel{\text{def}}{=} \bot \\
a(*\mathtt{X}) & \stackrel{\text{def}}{=} \neg s(\mathtt{X}) & s(*\mathtt{X}) & \stackrel{\text{def}}{=} \neg a(\mathtt{X}) \\
a(\mathtt{X} \circ \mathtt{Y}) & \stackrel{\text{def}}{=} a(\mathtt{X}) \wedge a(\mathtt{Y}) & s(\mathtt{X} \circ \mathtt{Y}) & \stackrel{\text{def}}{=} s(\mathtt{X}) \vee s(\mathtt{Y}) \\
a(\bullet\mathtt{X}) & \stackrel{\text{def}}{=} \Diamond^{-} a(\mathtt{X}) & s(\bullet\mathtt{X}) & \stackrel{\text{def}}{=} \Box s(\mathtt{X})
\end{array}$$

The modality $\Diamond^{-}$ is the backward existential modality associated with $\Box$. That is, as is usual with **DL**, we extend the language by adding the unary modal operator $\Box^{-}$. We write $\mathtt{L}^{+}$ [resp. $\mathtt{L}^{+}\mathtt{F}$] to denote the extension of $\mathtt{L}$ [resp. $\mathtt{LF}$] obtained by adding the axiom schemes $\Box^{-}\mathtt{p} \Rightarrow (\Box^{-}(\mathtt{p} \Rightarrow \mathtt{q}) \Rightarrow \Box^{-}\mathtt{q})$, $\mathtt{q} \Rightarrow \Box\Diamond^{-}\mathtt{q}$, $\mathtt{q} \Rightarrow \Box^{-}\Diamond\mathtt{q}$ and the necessitation rule: from ϕ infer $\Box^{-}\phi$. The language is extended appropriately by adding $\Box^{-}$, and $\Diamond^{-}\phi$ is defined as $\neg\Box^{-}\neg\phi$.

Theorem 3. *If* $\mathtt{X} \vdash \mathtt{Y}$ *is derivable in* $\delta\mathtt{LF}$*, then* $a(\mathtt{X}) \Rightarrow s(\mathtt{Y}) \in \mathtt{L}^{+}\mathtt{F}$*.*

The maps a and s can be found for instance in [Kra96] where they are called τ_1 and τ_2 respectively. The maps a and s give an intuitive way to interpret the meaning of the structural connectives depending on the polarity of their occurrence (either as antecedent part or as succedent part).

Lemma 1. *The following rules are admissible in* $\delta\mathtt{LF}$*:*

$$\frac{\phi_1 \wedge \phi_2 \vdash \mathtt{X}}{\phi_1 \circ \phi_2 \vdash \mathtt{X}}\ (\circ\vdash) \qquad \frac{\mathtt{X} \vdash \phi_1 \Rightarrow \phi_2}{\mathtt{X} \vdash *\phi_1 \circ \phi_2}\ (adm1) \qquad \frac{\mathtt{X} \vdash \phi_1 \vee \phi_2}{\mathtt{X} \vdash \phi_1 \circ \phi_2}\ (\vdash\circ)$$

$$\frac{\neg\phi \vdash \mathtt{X}}{*\phi \vdash \mathtt{X}}\ (*\vdash) \qquad \frac{\mathtt{F}(\phi) \vdash \mathtt{X}}{m(\mathtt{F}(\phi), 0) \vdash \mathtt{X}}\ (adm2) \qquad \frac{\mathtt{X} \vdash \neg\phi}{\mathtt{X} \vdash *\phi}\ (\vdash *)$$

Moreover, for each of these rules, if the premiss has a cut-free proof in $\delta\mathtt{LF}$*, then the conclusion also has a cut-free proof in* $\delta\mathtt{LF}$*.*

The proof of admissibility of the rules $(\vdash\circ)$, $(adm1)$, $(\circ\vdash)$, $(*\vdash)$ and $(\vdash *)$ is similar to [Kra96, Lemma 9]. Admissibility of $(adm2)$ is a mere consequence of the admissibility of the above rules.

Lemma 2. $\phi \vdash \phi$ *is cut-free derivable in* $\delta\mathtt{LF}$ *for any formula* ϕ*.*

Lemma 2 requires induction on the formation of ϕ. Theorem 4 is the **DL** version of Theorem 1 in [Avr84] for Gentzen-style calculi.

Theorem 4. *A formula* $\phi \in \mathtt{LF}$ *iff* $I \vdash \phi$ *is derivable in* $\delta\mathtt{LF}$*.*

Proof. The right to left direction is just an instance of Theorem 3. The left to right direction *requires* uses of the cut rule (to simulate the application of the *modus ponens* rule) and proceeds by induction on the length of the derivation in $\mathtt{LF}$ (viewed as an Hilbert-style system). Most of the cases can be found in [Wan94,Kra96,Wan98]. It remains to show that $I \vdash \Box(\mathtt{F}(\phi) \Rightarrow \phi) \Rightarrow \Box\phi$ has a proof in $\delta\mathtt{LF}$ which is done below using the fact that $\mathtt{F}(\phi) \vdash \mathtt{F}(\phi)$ and $\phi \vdash \phi$ are derivable in $\delta\mathtt{LF}$ by Lemma 2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\phi \vdash \phi \qquad \dfrac{\mathtt{F}(\phi) \vdash \mathtt{F}(\phi)}{m(\mathtt{F}(\phi), 0) \vdash \mathtt{F}(\phi)}\ (adm2)}{\mathtt{F}(\phi) \Rightarrow \phi \vdash *m(\mathtt{F}(\phi), 0) \circ \phi}\ (\Rightarrow\vdash)}{\Box(\mathtt{F}(\phi) \Rightarrow \phi) \vdash \bullet(*m(\mathtt{F}(\phi), 0) \circ \phi)}\ (\Box\vdash)}{\Box(\mathtt{F}(\phi) \Rightarrow \phi) \vdash \Box\phi}\ (\vdash\Box_{\mathtt{LF}})}{I \circ \Box(\mathtt{F}(\phi) \Rightarrow \phi) \vdash \Box\phi}\ (I_l)}{I \vdash \Box(\mathtt{F}(\phi) \Rightarrow \phi) \Rightarrow \Box\phi}\ (\vdash\Rightarrow)$$

As stated previously, any display calculus satisfying the conditions (C2)-(C8) from [Bel82] admits cut-elimination. Unfortunately $\delta\mathtt{LF}$ does not satisfy (C8) (see the appendix).

Specifically, the cut instance below breaks (C8):

$$\dfrac{\dfrac{\mathtt{X} \vdash \bullet(*m(\mathtt{F}(\phi),0)\circ\phi)}{\mathtt{X}\vdash\Box\phi}\,(\vdash\Box_{\mathtt{LF}}) \qquad \dfrac{\phi\vdash\mathtt{Y}}{\Box\phi\vdash\bullet\mathtt{Y}}\,(\Box\vdash)}{\mathtt{X}\vdash\bullet\mathtt{Y}}\,(cut)$$

where there is a formula ψ in $m(\mathtt{F}(\phi),0)$ that is not a subformula of ϕ. For instance, such cases are easy to find with the display calculi δG and δGrz. Furthermore, in order to infer $\mathtt{X}\vdash\bullet\mathtt{Y}$ from $\mathtt{X}\vdash\bullet(*m(\mathtt{F}(\phi),0)\circ\phi)$ and $\phi\vdash\mathtt{Y}$, no cut can be used on ψ if (C8) has to be satisfied. In the display calculus $\delta\mathtt{LF}$, for all the derivations of the sequent $\mathtt{X}''\vdash\mathtt{Y}''$ from $\mathtt{X}\vdash\bullet(*m(\mathtt{F}(\phi),0)\circ\phi)$, if a cut with a cut-formula that is not a subformula of ϕ is forbidden, then either $\mathtt{X}''\vdash\mathtt{Y}''$ contains ψ as the subformula of some formula/structure (see the introduction rules different from $(\vdash\Box_{\mathtt{LF}})$), or $\mathtt{X}''\vdash\mathtt{Y}''$ contains $\Box\phi$ as the subformula of some formula/structure (see the $(\vdash\Box_{\mathtt{LF}})$-rule). So, *in the general case*, there is no way to derive $\mathtt{X}\vdash\bullet\mathtt{Y}$ since neither ψ nor $\Box\phi$ are guaranteed to occur in it.

We say that $\mathtt{LF}$ is *pseudo displayable* iff for any $\phi\in\mathtt{FML}$, $I\vdash\phi$ has a proof in $\delta\mathtt{LF}$ iff $I\vdash\phi$ has a cut-free proof in $\delta\mathtt{LF}$. "Pseudo" because strong cut-elimination is couched in terms of arbitrary sequents $\mathtt{X}\vdash\mathtt{Y}$ rather than sequents of the form $I\vdash\phi$. For mechanisation "pseudo" is sufficient for our needs since we want to check whether $\phi\in\mathtt{LF}$. We now provide a characterisation of the class of pseudo displayable logics and show that both G and Grz are pseudo displayable.

4 Transformations from LF into L

In this section, $\mathtt{L}$ is a properly displayable logic and $\mathtt{F}$ is a formula generation map. Let $f:\mathtt{FML}\times\{0,1\}\to\mathtt{FML}$ be the following map for $i\in\{0,1\}$:

for any $\mathtt{p}\in\mathtt{PRP}$, $f(\mathtt{p},0)\overset{\text{def}}{=}f(\mathtt{p},1)\overset{\text{def}}{=}\mathtt{p}$ $\quad f(\top,i)\overset{\text{def}}{=}\top$ $\quad f(\bot,i)\overset{\text{def}}{=}\bot$
$f(\phi_1\oplus\phi_2,i)\overset{\text{def}}{=}f(\phi_1,i)\oplus f(\phi_2,i)$ for $\oplus\in\{\wedge,\vee\}$
$f(\neg\phi,0)\overset{\text{def}}{=}\neg f(\phi,1)$ $\quad f(\neg\phi,1)\overset{\text{def}}{=}\neg f(\phi,0)$
$f(\phi_1\Rightarrow\phi_2,1)\overset{\text{def}}{=}f(\phi_1,0)\Rightarrow f(\phi_2,1)$ $\quad f(\phi_1\Rightarrow\phi_2,0)\overset{\text{def}}{=}f(\phi_1,1)\Rightarrow f(\phi_2,0)$
$f(\Box\phi,1)\overset{\text{def}}{=}\Box(f(\mathtt{F}(\phi),0)\Rightarrow f(\phi,1))$ $\quad f(\Box\phi,0)\overset{\text{def}}{=}\Box f(\phi,0)$

In $f(\phi,i)$, the index i carries information about the *polarity* of ϕ in the translation process as in [BH94]. The map f also generalises one of the maps from G into K4 defined in [BH94]. By simultaneous induction one can show that for any $\phi\in\mathtt{FML}$ and for any $i\in\{0,1\}$, $\phi\Leftrightarrow f(\phi,i)\in\mathtt{LF}$. Moreover, for any $\phi\in\mathtt{FML}$: $f(\phi,0)\Rightarrow\phi\in\mathtt{L}$, $\phi\Rightarrow f(\phi,1)\in\mathtt{L}$ and therefore $f(\phi,0)\Rightarrow f(\phi,1)\in\mathtt{L}$.

Lemma 3. *Every positive [resp. negative] occurrence of*

1. *$\Box\psi$ in $f(\phi,1)$ is of the form $\Box(f(\mathtt{F}(\varphi),0)\Rightarrow f(\varphi,1))$ [resp. $\Box f(\varphi,0)$] for some subformula φ of ϕ;*

2. *$\neg\psi$ in $f(\phi,1)$ is of the form $\neg f(\varphi,0)$ [resp. $\neg f(\varphi,1)$] for some subformula φ of ϕ;*
3. *$\psi_1 \Rightarrow \psi_2$ in $f(\phi,1)$ is of the form $f(\varphi_1,0) \Rightarrow f(\varphi_2,1)$ [resp. $f(\varphi_1,1) \Rightarrow f(\varphi_2,0)$] for some subformulae φ_1, φ_2 of ϕ;*
4. *$\psi_1 \oplus \psi_2$ ($\oplus \in \{\wedge,\vee\}$) in $f(\phi,1)$ is of the form $f(\varphi_1,1) \oplus f(\varphi_2,1)$ [resp. $f(\varphi_1,0) \oplus f(\varphi_2,0)$] for some subformulae φ_1, φ_2 of ϕ;*

The proof of Lemma 3 is by an easy verification. Lemma 3 is used in the proof of Theorem 5 below. We extend the map f to structures in the following way ($i \in \{0,1\}$):

$$f(I,i) \stackrel{\text{def}}{=} I \qquad f(*\mathtt{X},i) \stackrel{\text{def}}{=} *f(\mathtt{X},1-i)$$
$$f(\mathtt{X}\circ\mathtt{Y},i) \stackrel{\text{def}}{=} f(\mathtt{X},i)\circ f(\mathtt{Y},i) \qquad f(\bullet\mathtt{X},i) \stackrel{\text{def}}{=} \bullet f(\mathtt{X},i)$$

By induction on the structure of ϕ, the rule below is admissible in $\delta\mathtt{L}$:

$$\frac{f(m(\phi,0),0) \vdash \mathtt{X}}{f(\phi,0) \vdash \mathtt{X}}\ (adm3)$$

Theorem 5 below is the main result of the paper.

Theorem 5. *The statements below are equivalent:*

1. *For all formulae ϕ, $\phi \in \mathtt{LF}$ iff $f(\phi,1) \in \mathtt{L}$.*
2. *$\mathtt{LF}$ is pseudo displayable.*

That is, (weak) cut-elimination of $\delta\mathtt{LF}$ is equivalent to the theoremhood preserving nature of f from $\mathtt{LF}$ into $\mathtt{L}$. Its proof is purely syntactic and therefore it does *not depend on the class of modal frames that possibly characterises* $\mathtt{LF}$. Moreover, Theorem 5 goes beyond the mechanisation aspect since it provides a characterisation of the class of pseudo displayable logics which is not directly based on a cut-elimination procedure.

The proof of Theorem 5 is long and tedious. For instance, when (2) holds and $\phi \in \mathtt{LF}$, to show that $f(\phi,1) \in \mathtt{L}$, we show that in any cut-free proof Π of $I \vdash \phi$ in $\delta\mathtt{LF}$, for any sequent $\mathtt{X} \vdash \mathtt{Y}$ in Π, $f(\mathtt{X},0) \vdash f(\mathtt{Y},1)$ has a cut-free proof in $\delta\mathtt{L}$. By way of example, the proof step (in $\delta\mathtt{LF}$) shown below left is transformed into the proof steps (in $\delta\mathtt{L}$) shown below right:

$$\frac{\mathtt{X} \vdash \bullet(*m(\mathtt{F}(\psi),0)\circ\psi)}{\mathtt{X} \vdash \Box\psi}\ (\vdash \Box_{\mathtt{LF}})$$

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{f(\mathtt{X},0) \vdash \bullet(*f(m(\mathtt{F}(\psi),0),0)\circ f(\psi,1))}{\bullet f(\mathtt{X},0) \vdash *f(m(\mathtt{F}(\psi),0),0)\circ f(\psi,1)}\ (dp)}{\bullet f(\mathtt{X},0) \vdash f(\psi,1)\circ *f(m(\mathtt{F}(\psi),0),0)}\ (com_r)}{f(m(\mathtt{F}(\psi),0),0) \vdash *\bullet f(\mathtt{X},0)\circ f(\psi,1)}\ (dp)}{f(\mathtt{F}(\psi),0) \vdash *\bullet f(\mathtt{X},0)\circ f(\psi,1)}\ (adm3)}{\bullet f(\mathtt{X},0)\circ f(\mathtt{F}(\psi),0) \vdash f(\psi,1)}\ (dp)}{\bullet f(\mathtt{X},0) \vdash f(\mathtt{F}(\psi),0) \Rightarrow f(\psi,1)}\ (\vdash\Rightarrow)}{f(\mathtt{X},0) \vdash \bullet(f(\mathtt{F}(\psi),0) \Rightarrow f(\psi,1))}\ (dp)}{f(\mathtt{X},0) \vdash \Box(f(\mathtt{F}(\psi),0) \Rightarrow f(\psi,1))}\ (\vdash \Box_{\mathtt{L}})$$

$f^{-1}(I, i) \overset{\text{def}}{=} I \qquad f^{-1}(\top, i) \overset{\text{def}}{=} \top \qquad f^{-1}(\bot, i) \overset{\text{def}}{=} \bot$

$f^{-1}(\mathtt{X} \circ \mathtt{Y}, i) \overset{\text{def}}{=} f^{-1}(\mathtt{X}, i) \circ f^{-1}(\mathtt{Y}, i)$ or undefined

$f^{-1}(\bullet\mathtt{X}, i) \overset{\text{def}}{=} \bullet f^{-1}(\mathtt{X}, i)$ or undefined

$f^{-1}(*\mathtt{X}, 1-i) \overset{\text{def}}{=} *f^{-1}(\mathtt{X}, i)$ or undefined

$f^{-1}(\mathtt{p}, i) \overset{\text{def}}{=} \mathtt{p}$ for any $\mathtt{p} \in \mathtt{PRP}$

for $\oplus \in \{\wedge, \vee\}$, $f^{-1}(\phi \oplus \psi, i) \overset{\text{def}}{=} f^{-1}(\phi, i) \oplus f^{-1}(\psi, i)$ or undefined

$f^{-1}(\phi \Rightarrow \psi, 1) \overset{\text{def}}{=} f^{-1}(\phi, 0) \Rightarrow f^{-1}(\psi, 1)$ or undefined

$f^{-1}(\phi \Rightarrow \psi, 0) \overset{\text{def}}{=} f^{-1}(\phi, 1) \Rightarrow f^{-1}(\psi, 0)$ or undefined

$f^{-1}(\neg\phi, 1-i) \overset{\text{def}}{=} \neg f^{-1}(\phi, i)$ or undefined

$f^{-1}(\Box\phi, 0) \overset{\text{def}}{=} \Box f^{-1}(\phi, 0)$ or undefined

$$f^{-1}(\Box\phi, 1) \overset{\text{def}}{=} \begin{cases} \Box f^{-1}(\phi_2, 1) \text{ if } \phi = (\phi_1 \Rightarrow \phi_2) \text{ and } f^{-1}(\phi_2, 1) \text{ is defined} \\ \text{undefined otherwise} \end{cases}$$

where "$x \overset{\text{def}}{=} y$ or undefined" means: $x \overset{\text{def}}{=} \begin{cases} y \text{ if all components of } y \text{ are defined} \\ \text{undefined otherwise} \end{cases}$

Fig. 5. Definition of $f^{-1}(.)$ for $i \in \{0, 1\}$.

When (1) holds, $f(\phi, 1) \in \mathtt{L}$ implies $I \vdash f(\phi, 1)$ has a cut-free proof Π in $\delta\mathtt{L}$, and $\phi \in \mathtt{LF}$ implies $I \vdash \phi$ is provable in $\delta\mathtt{LF}$. To show that $I \vdash \phi$ has a *cut-free* proof in $\delta\mathtt{LF}$, we prove that for every sequent $\mathtt{X} \vdash \mathtt{Y}$ in Π, the sequent $f^{-1}(\mathtt{X}, 0) \vdash f^{-1}(\mathtt{Y}, 1)$ admits a cut-free proof in $\delta\mathtt{LF}$ with the partial function f^{-1} defined in Figure 5. Since $\delta\mathtt{L}$ satisfies (C1)-(C8) (see the appendix), for every sequent $\mathtt{X} \vdash \mathtt{Y}$ in Π, both $f^{-1}(\mathtt{X}, 0)$ and $f^{-1}(\mathtt{Y}, 1)$ are defined (Lemma 3 is also used to show this property). Because the map $f^{-1} : \mathtt{struc}(\delta\mathtt{L}) \times \{0, 1\} \rightarrow \mathtt{struc}(\delta\mathtt{L})$ satisfies $f^{-1}(f(\phi, 1), 1) = \phi$ and $f^{-1}(f(\phi, 0), 0) = \phi$, the end-sequent $I \vdash f(\phi, 1)$ of Π becomes $I \vdash \phi$ in this procedure, as desired.

The proof of Theorem 5 actually shows that **DL** is particularly well-designed to reason about polarity, succedent and antecedent parts. One of the translations from G into K4 in [BH94] is exactly the map f when $\mathtt{L}$ is K4 and $\mathtt{F}$ is $\mathtt{F}_G$.

Corollary 1. *G is pseudo displayable.*

Let $\delta^- G$ be the calculus δG minus the cut-rule. Thus $\delta^- G$ satisfies *all* the conditions (C1)-(C8) and for any $\phi \in \mathtt{FML}$, $\phi \in G$ iff $I \vdash \phi$ has a proof in $\delta^- G$. At first glance, this seems to contradict the fact that G is not properly displayable in the sense of [Kra96]. However, in [Kra96], all the axioms added to K are transformed into structural rules. By contrast, in $\delta^- G$, one of the axioms is encoded in the $(\vdash \Box_G)$-rule. This opens an avenue to define display calculi satisfying (C1)-(C8) for modal logics that are not properly displayable.

Theorem 6. *For every pseudo displayable logic* $\mathtt{LF}$*, there is an* $\mathcal{O}(n^3.log\ n)$*-time transformation* g *such that any* $\phi \in \mathtt{LF}$ *iff* $g(\phi) \in \mathtt{L}$.

Form of ϕ_i	ψ_i
$\top$	$(\bigwedge_{i=0}^{md(\phi)} \Box^i(\mathtt{p}_{i,1} \Leftrightarrow \top) \wedge \Box^i(\mathtt{p}_{i,0} \Leftrightarrow \top))$
$\bot$	$(\bigwedge_{i=0}^{md(\phi)} \Box^i(\mathtt{p}_{i,1} \Leftrightarrow \bot) \wedge \Box^i(\mathtt{p}_{i,0} \Leftrightarrow \bot))$
$\mathtt{p}$	$(\bigwedge_{i=0}^{md(\phi)} \Box^i(\mathtt{p}_{i,0} \Leftrightarrow \mathtt{p}_{i,1}))$
$\neg\phi_j$	$(\bigwedge_{i=0}^{md(\phi)} \Box^i(\mathtt{p}_{i,1} \Leftrightarrow \neg\mathtt{p}_{j,0}) \wedge \Box^i(\mathtt{p}_{i,0} \Leftrightarrow \neg\mathtt{p}_{j,1}))$
$\phi_{i_1} \wedge \phi_{i_2}$	$(\bigwedge_{i=0}^{md(\phi)} \Box^i(\mathtt{p}_{i,1} \Leftrightarrow (\mathtt{p}_{i_1,1} \wedge \mathtt{p}_{i_2,1})) \wedge \Box^i(\mathtt{p}_{i,0} \Leftrightarrow (\mathtt{p}_{i_1,0} \wedge \mathtt{p}_{i_2,0})))$
$\phi_{i_1} \vee \phi_{i_2}$	$(\bigwedge_{i=0}^{md(\phi)} \Box^i(\mathtt{p}_{i,1} \Leftrightarrow (\mathtt{p}_{i_1,1} \vee \mathtt{p}_{i_2,1})) \wedge \Box^i(\mathtt{p}_{i,0} \Leftrightarrow (\mathtt{p}_{i_1,0} \vee \mathtt{p}_{i_2,0})))$
$\phi_{i_1} \Rightarrow \phi_{i_2}$	$(\bigwedge_{i=0}^{md(\phi)} \Box^i(\mathtt{p}_{i,1} \Leftrightarrow (\mathtt{p}_{i_1,0} \Rightarrow \mathtt{p}_{i_2,1})) \wedge \Box(\mathtt{p}_{i,0} \Leftrightarrow (\mathtt{p}_{i_1,1} \Rightarrow \mathtt{p}_{i_2,0})))$
$\Box\phi_j$	$(\bigwedge_{i=0}^{md(\phi)} \Box^i(\mathtt{p}_{i,1} \Leftrightarrow \Box(\mathtt{F}'(\mathtt{p}_{j,0}, \mathtt{p}_{j,1}) \Rightarrow \mathtt{p}_{j,1})) \wedge \Box^i(\mathtt{p}_{i,0} \Leftrightarrow \Box\mathtt{p}_{j,0}))$

Fig. 6. Definition of ψ_i

The right-hand side of the definition of $f(\Box\psi, 1)$ may require several calls to $f(\psi, 0)$ and $f(\psi, 1)$, so f is not necessarily computable in $\mathcal{O}(n^3.log\ n)$-time. However, we can use a variant of f using a standard renaming technique (see e.g. [Min88]). Let $md(\phi)$ denote the *modal depth* of ϕ, let $\Box^0\varphi \overset{\text{def}}{=} \varphi$ and $\Box^{i+1}\varphi \overset{\text{def}}{=} \Box\Box^i\varphi$. Then for any extension L of K, $\phi \in \mathtt{L}$ iff $(\bigwedge_{i=0}^{md(\phi)} \Box^i(\mathtt{p}_{new} \Leftrightarrow \psi)) \Rightarrow \phi' \in \mathtt{L}$ where ϕ' is obtained by replacing every occurrence of ψ in ϕ by $\mathtt{p}_{new}$, a new propositional variable not occurring in ϕ. When $\mathtt{K4} \subseteq \mathtt{L}$, $\phi \in \mathtt{L}$ iff $(\Box(\mathtt{p}_{new} \Leftrightarrow \psi) \wedge (\mathtt{p}_{new} \Leftrightarrow \psi)) \Rightarrow \phi' \in \mathtt{L}$.

Proof. (Theorem 6) The key point to define g is to observe that there is a map $\mathtt{F}' : \mathtt{FML} \times \mathtt{FML} \rightarrow \mathtt{FML}$ and a formula $\psi_{\mathtt{F}'}$ containing at most *two* atomic propositions, say p and q, such that

- $\mathtt{F}'(\varphi_1, \varphi_2)$ is obtained from $\psi_{\mathtt{F}'}$ by replacing simultaneously every occurrence of p [resp. q] by φ_1 [resp. φ_2];
- for any $\varphi \in \mathtt{FML}$, $f(\mathtt{F}(\varphi), 0) = \mathtt{F}'(f(\varphi, 0), f(\varphi, 1))$.

For instance, if $\mathtt{F} = \lambda\mathtt{p}.\mathtt{p} \wedge \neg\mathtt{p}$ then $\mathtt{F}' = \lambda\mathtt{pq}.\mathtt{p} \wedge \neg\mathtt{q}$.

Let ϕ be a modal formula we wish to translate from LF into L. Let $\phi_1, \ldots, \phi_m$ be an enumeration (without repetition) of all subformulae of ϕ, in increasing order of size. We shall build a formula $g(\phi)$ using the set $\{\mathtt{p}_{i,j} : 1 \leq i \leq m,\ j \in \{0, 1\}\}$ of atomic propositions such that $g(\phi) \in \mathtt{L}$ iff $f(\phi, 1) \in \mathtt{L}$. We could also just consider the set $\{\mathtt{p}_i : i \in \omega\}$ of atomic propositions and use a 1-1 mapping from $\omega^2 \rightarrow \omega$, but for simplicity, the present option is the most convenient.

Moreover, $g(\phi)$ can be computed in time $\mathcal{O}(|\phi|^3.log\ |\phi|)$. For $i \in \{1, \ldots, m\}$, we create a formula ψ_i as shown in Figure 6 and let $g(\phi) \overset{\text{def}}{=} (\bigwedge_{i=1}^{m} \psi_i) \Rightarrow \mathtt{p}_{m,1}$. If $\mathrm{K4} \subseteq \mathtt{L}$, the generalised conjunction in Figure 6 is needed only for $i \in \{0, 1\}$.

Each $|\psi_i|$ is in $\mathcal{O}(|\phi|^2)$ since $\Sigma_{i=0}^{md(\phi)} i$ is in $\mathcal{O}(|\phi|^2)$. So $|g(\phi)|$ is in $\mathcal{O}(|\phi| \times (|\phi|^2 \times log\ |\phi|))$. As usual in complexity theory, the extra $log\ |\phi|$ factor in the size of ϕ is because we need an index of size $\mathcal{O}(log\ |\phi|)$ for these different atomic propositions. That is, these indices are represented in binary writing.

The map g is a generalisation of the map from Grz into S4 defined in [DG99a]. One of the maps from G into K4 in [BH94] is linear-time and does not use renaming (which allows us to treat the general case). We are currently investigating if their map can be generalised by considering the map $f' : \mathtt{FML} \times \{0,1\} \rightarrow \mathtt{FML}$ inductively defined as f except that $f'(\Box\phi, 1) \overset{\text{def}}{=} \Box(\mathtt{F}(\phi) \Rightarrow f'(\phi, 1))$. Another map in **P** from G into K4 is given in [Fit83, Chapter 5].

5 Another Characterisation

Theorem 5 has a counterpart when LF has a traditional Gentzen system although additional conditions are required. In this section, L is a properly displayable logic and F is a formula generation map.

Definition 2. *Assume* L*, properly displayable by assumption, has a traditional Gentzen system* GL *in which*

1. GL *extends a standard Gentzen system for PC containing contraction, weakening, exchange and cut (shown below left), and* GL *satisfies cut-elimination;*
2. *any sequent $\Gamma \vdash \Delta$ is derivable in* GL *iff $(\bigwedge_{\phi\in\Gamma} \phi) \Rightarrow (\bigvee_{\phi\in\Delta} \phi) \in \mathtt{L}$;*
3. *the $(\Box \vdash)$-rule (if any) and the $(\vdash \Box)$-rule have the form*

$$\frac{\Gamma, \phi \vdash \Delta}{\Gamma, \Box\phi \vdash \Delta}\ (\Box \vdash) \qquad\qquad \frac{\Sigma' \vdash \phi}{\Sigma \vdash \Box\phi}\ (\vdash \Box)$$

where there is a map $h : \mathtt{FML} \rightarrow \mathtt{FML}$ such that $h(\Sigma) = \Sigma'$, $h(f(\Sigma, 0)) = f(\Sigma', 0)$ for any sequence Σ to which the $(\vdash \Box)$-rule is applicable. Both h and f (defined via F*) are naturally extended to sequences. Moreover, $f(\Sigma, 0)$ satisfies any further conditions on the $(\vdash \Box)$-rule iff Σ does too: for example, that Σ must contain only formulae beginning with $\Box$.*

These sequents consist of comma-separated lists of formulae. Then, LF *is* pseudo Gentzenisable *iff the Gentzen system* GLF *obtained from* GL *by replacing the $(\vdash \Box)$-rule by $(\vdash \Box_{\mathtt{LF}})$ shown below right enjoys cut-elimination.*

$$\frac{\Gamma \vdash \phi, \Delta \quad \Gamma', \phi \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ (cut) \qquad\qquad \frac{\mathtt{F}(\phi), \Sigma' \vdash \phi}{\Sigma \vdash \Box\phi}\ (\vdash \Box_{\mathtt{LF}})$$

Definition 2(3.) ensures that (C4) is satisfied when the *structures* are simply sequences of formulae. Following [Avr84, Theorem 1], we can show that GLF is sound and complete for LF. Here is the Gentzen counterpart of Theorem 5.

Theorem 7. *If* L *satisfies assumptions (1)-(3) from Definition 2 then the statements below are equivalent:*

1. *For all $\phi \in \mathtt{FML}$, $\phi \in \mathtt{LF}$ iff $f(\phi, 1) \in \mathtt{L}$.*
2. LF *is pseudo Gentzenisable.*

By [Avr84, Corollary 3.1], Grz is pseudo Gentzenisable and therefore Grz is pseudo displayable. So by Theorem 7, for any $\phi \in \mathtt{FML}$, $\phi \in Grz$ iff $f(\phi, 1) \in S4$ where f is defined with $\mathtt{F} = \mathtt{F}_{Grz}$. See [DG99a] for a more detailed case-study showing how to translate Grz into a *decidable* subset of first-order logic.

6 Concluding Remarks

We can summarise the general situation as follows:

Theorem 8. *Assume logic* $\mathtt{L}$ *is properly displayable and* $\mathtt{F}$ *is a formula generation map. Then,*

(I) *(a)* $\mathtt{LF}$ *is pseudo displayable iff*
(b) for all $\phi \in \mathtt{FML}$, $\phi \in \mathtt{LF}$ *iff* $f(\phi, 1) \in \mathtt{L}$ *(where* f *is defined via* $\mathtt{F}$*).*
(II) *If* $\mathtt{L}$ *satisfies the assumptions (1)-(3) from Definition 2, then (b) holds iff* $\mathtt{LF}$ *is pseudo Gentzenisable.*
(III) *When (a) holds [resp. and when K4* $\subseteq \mathtt{L}$*], there is an* $\mathcal{O}(n^3.log\ n)$*-time [resp. an* $\mathcal{O}(n.log\ n)$*-time] transformation from* $\mathtt{LF}$ *into* $\mathtt{L}$*.*

Our contribution can be summarised as follows:

1. Automated Reasoning: We gave a uniform framework to translate every pseudo-displayable modal logic $\mathtt{LF}$, which may not be first-order definable, into the first-order definable primitive modal logic $\mathtt{L}$. The transformations are at most in $\mathcal{O}(n^3.log\ n)$-time. Since a linear-time transformation from each $\mathtt{L}$ into FOL is known, we can use classical theorem provers for any pseudo-displayable logic $\mathtt{LF}$.
2. Cut-elimination: (Often a desirable property for mechanisation) Although the calculi $\delta\mathtt{LF}$ do not satisfy condition (C8), we can nevertheless characterise weak cut-elimination by the theoremhood preserving nature of f.
3. Display Logic: We defined **DL** calculi for logics that are not properly displayable à la [Kra96].

References

AEH90. Y. Auffray, P. Enjalbert, and J.-J. Herbrard. Strategies for modal resolution: results and problems. *Journal of Automated Reasoning*, 6:1–38, 1990.

Avr84. A. Avron. On modal systems having arithmetical interpretations. *Journal of Symbolic Logic*, 49(3):935–942, 1984.

Bel82. N. Belnap. Display logic. *Journal of Philosophical Logic*, 11:375–417, 1982.

BG86. M. Borga and P. Gentilini. On the proof theory of the modal logic Grz. *Zeitschrift für Mathematik Logik und Grundlagen der Mathematik*, 32:145–148, 1986.

BH94. Ph. Balbiani and A. Herzig. A translation from the modal logic of provability into K4. *Journal of Applied Non-Classical Logics*, 4:73–77, 1994.

Boo93. G. Boolos. *The Logic of Provability.* Cambridge University Press, 1993.

DG99a. S. Demri and R. Goré. An $\mathcal{O}((n.log\ n)^3)$-time transformation from Grz into decidable fragments of classical first-order logic. In *Selected papers of FTP'98, Vienna.* LNAI, Springer-Verlag, 1999. to appear.

DG99b. S. Demri and R. Goré. Theoremhood preserving maps as a characterisation of cut elimination for provability logics. Technical report, A.R.P., A.N.U., 1999. Forthcoming.

dMP95. G. d'Agostino, A. Montanari, and A. Policriti. A set-theoretical translation method for polymodal logics. *Journal of Automated Reasoning*, 15:317–337, 1995.

Fit83. M. Fitting. *Proof methods for modal and intuitionistic logics.* D. Reidel Publishing Co., 1983.

GHM98. H. Ganzinger, U. Hustadt, and R. Meyer, C. Schmidt. A resolution-based decision procedure for extensions of K4. In *2nd Workshop on Advances in Modal Logic (AiML'98), Uppsala, Sweden*, October 1998. to appear.

Gor99. R. Goré. Tableaux methods for modal and temporal logics. In M. d'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableaux Methods.* Kluwer, Dordrecht, 1999. To appear.

Her89. A. Herzig. *Raisonnement automatique en logique modale et algorithmes d'unification.* PhD thesis, Université P. Sabatier, Toulouse, 1989.

HS97. U. Hustadt and R. Schmidt. On evaluating decision procedures for modal logic. In *IJCAI-15*, pages 202–207. Morgan Kaufmann, 1997.

Kra96. M. Kracht. Power and weakness of the modal display calculus. In H. Wansing, editor, *Proof theory of modal logic*, pages 93–121. Kluwer Academic Publishers, 1996.

Lei81. D. Leivant. On the proof theory of the modal logic for arithmetical provability. *Journal of Symbolic Logic*, 46(3):531–538, 1981.

Mas94. F. Massacci. Strongly analytic tableaux for normal modal logics. In A. Bundy, editor, *CADE-12*, pages 723–737. LNAI 814, July 1994.

Min88. G. Mints. Gentzen-type and resolution rules part I: propositional logic. In P. Martin-Löf and G. Mints, editors, *International Conference on Computer Logic, Tallinn*, pages 198–231. LNCS 417, 1988.

Mor76. Ch. Morgan. Methods for automated theorem proving in non classical logics. *IEEE Transactions on Computers*, 25(8):852–862, 1976.

NS98. A. Nonnengart and A. Szalas. A fixpoint approach to second-order quantifier elimination with applications to correspondence theory. In E. Orłowska, editor, *Logic at Work. Essays Dedicated to the Memory of Helena Rasiowa*, pages 89–108. Physica Verlag, 1998.

Ohl88. H.J. Ohlbach. A resolution calculus for modal logics. In *CADE-9*, pages 500–516. LNCS 310, 1988.

Ohl98. H.J. Ohlbach. Combining Hilbert style and semantic reasoning in a resolution framework. In C. Kirchner and H. Kirchner, editors, *CADE-15, Lindau, Germany*, pages 205–219. LNAI 1421, 1998.

Pap94. Ch. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

Sah75. H. Sahlqvist. Completeness and correspondence in the first and second order semantics for modal logics. In S. Kanger, editor, *3rd Scandinavian Logic Symposium*, pages 110–143. North Holland, 1975.

Sch99. R. Schmidt. Decidability by resolution for propositional modal logics. *Journal of Automated Reasoning*, 1999. To appear.

SV80. G. Sambin and S. Valentini. A modal sequent calculus for a fragment of arithmetic. *Studia Logica*, 39:245–256, 1980.

SV82. G. Sambin and S. Valentini. The modal logic of provability . The sequential approach. *Journal of Philosophical Logic*, 11:311–342, 1982.

Val83. S. Valentini. The modal logic of provability: cut-elimination. *Journal of Philosophical Logic*, 12:471–476, 1983.

Wan94. H. Wansing. Sequent calculi for normal modal propositional logics. *Journal of Logic and Computation*, 4(2):125–142, 1994.

Wan98. H. Wansing. *Displaying Modal Logic*, volume 3 of *Trends in Logic.* Kluwer Academic Publishers, Dordrecht, 1998.

Appendix: Belnap's Conditions.

For every sequent rule Belnap [Bel82, page 388] first defines the following notions: in an application *Inf* of a sequent rule (ρ), "constituents occurring as part of occurrences of structures assigned to structure-variables are defined to be **parameters** of *Inf* ; all other constituents are defined as **nonparametric**, including those assigned to formula-variables. Constituents occupying similar positions in occurrences of structures assigned to the same structure-variable are defined as **congruent** in *Inf* ". The eight (actually seven) conditions shown below are from [Kra96] and [Wan98]:

(C1) Each formula which is a constituent of some premiss of a rule ρ is a subformula of some formula in the conclusion of ρ.
(C2) Congruent parameters are occurrences of the same structure.
(C3) Each parameter is congruent to at most one constituent in the conclusion. Equivalently, no two constituents of the conclusion are congruent to each other.
(C4) Congruent parameters are either all antecedent parts or all succedent parts of their respective sequent.
(C5) If a formula is non-parametric in the conclusion of a rule ρ, it is either the entire antecedent, or the entire succedent. Such a formula is called a **principal** formula.
(C6/7) Each rule is closed under simultaneous substitution of arbitrary structures for congruent parameters.
(C8) If there are inferences $\mathcal{I}_1$ and $\mathcal{I}_2$ with respective conclusions $\mathtt{X} \vdash \varphi$ and $\varphi \vdash \mathtt{Y}$ with φ *principal* in both inferences, and if cut is applied to obtain $\mathtt{X} \vdash \mathtt{Y}$, then
 (i) either $\mathtt{X} \vdash \mathtt{Y}$ is identical to one of $\mathtt{X} \vdash \varphi$ and $\varphi \vdash \mathtt{Y}$;
 (ii) or there is derivation of $\mathtt{X} \vdash \mathtt{Y}$ from the premisses of $\mathcal{I}_1$ and $\mathcal{I}_2$ in which every cut-formula of any application of cut is a proper subformula of φ.

Decision Procedures for Guarded Logics

Erich Grädel

Mathematische Grundlagen der Informatik, RWTH Aachen
graedel@informatik.rwth-aachen.de

Abstract. Different variants of guarded logics (a powerful generalization of modal logics) are surveyed and the recent decidability result for guarded fixed point logic (obtained in joint work with I. Walukiewicz) is explained. The exposition given here emphasizes the tree model property of guarded logics: every satisfiable sentence has a model of bounded tree width. Based on the tree model property, different methods for reasoning with guarded fixed point sentences are presented:

(1) reduction to the monadic theory of countable trees (SωS);
(2) reduction to the μ-calculus with backwards modalities;
(3) the automata theoretic method (which gives theoretically optimal complexity results).

1 Introduction

Guarded logics are defined by restricting quantification in first-order logic, second-order logic, fixed point logics or infinitary logics in such a way that, semantically speaking, each subformula can refer only to tuples of elements that are 'very close together' or 'guarded'.

Syntactically this means that all first-order quantifiers must be relativised by certain 'guard formulae' that tie together all the free variables in the scope of the quantifier. Quantification is of the form

$$\exists \boldsymbol{y}(\alpha(\boldsymbol{x},\boldsymbol{y}) \land \psi(\boldsymbol{x},\boldsymbol{y})) \qquad \text{or} \qquad \forall \boldsymbol{y}(\alpha(\boldsymbol{x},\boldsymbol{y}) \rightarrow \psi(\boldsymbol{x},\boldsymbol{y}))$$

where quantifiers may range over a tuple $\boldsymbol{y}$ of variables, but are 'guarded' by a formula α that must contain all the free variables of the formula ψ that is quantified over. The guard formulae are of a simple syntactic form (in the basic version, they are just atoms). Depending on the conditions imposed on guard formulae, one has logics with different levels of 'closeness' or 'guardedness'. Again, there is a syntactic and a semantic view of such guard conditions.

Let us start with the logic GF, the *guarded fragment* of first-order logic, as it was introduced by Andréka, van Benthem and Nemeti [1].

Definition 1. GF is defined inductively as follows:

(1) Every relational atomic formula $Rx_{i_1} \cdots x_{i_m}$ or $x_i = x_j$ belongs to GF.
(2) GF is closed under boolean operations.

(3) If $\boldsymbol{x}, \boldsymbol{y}$ are tuples of variables, $\alpha(\boldsymbol{x}, \boldsymbol{y})$ is a positive atomic formula and $\psi(\boldsymbol{x}, \boldsymbol{y})$ is a formula in GF such that $\text{free}(\psi) \subseteq \text{free}(\alpha) = \boldsymbol{x} \cup \boldsymbol{y}$, then also the formulae

$$\exists \boldsymbol{y}(\alpha(\boldsymbol{x}, \boldsymbol{y}) \wedge \psi(\boldsymbol{x}, \boldsymbol{y})) \quad \text{and} \quad \forall \boldsymbol{y}(\alpha(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \psi(\boldsymbol{x}, \boldsymbol{y}))$$

belong to GF.

Here free(ψ) means the set of free variables of ψ. An atom $\alpha(\boldsymbol{x}, \boldsymbol{y})$ that relativizes a quantifier as in rule (3) is the *guard* of the quantifier. Hence in GF, guards must be atoms. But the really crucial property of guards (also for the more powerful guarded logics that we will consider below) is that it must contain all free variables of the formula that is quantified over.

The main motivation for introducing the guarded fragment was to explain and generalize the good algorithmic and model-theoretic properties of propositional *modal* logics. Recall that the basic (poly)modal logic ML (also called K_n) extends propositional logic by the possibility to construct formulae $\langle a \rangle \psi$ and $[a]\psi$ (for any a from a given set A of 'actions' or 'modalities') with the meaning that ψ holds at some, respectively each, a-successor of the current state. Although ML is formally a propositional logic we really view it as a fragment of first-order logic. Kripke structures which provide the semantics for modal logics are just relational structures with only unary and binary relations. There is a standard translation taking every formula $\psi \in$ ML to a first-order formula $\psi^*(x)$ with one free variable, such that for every Kripke structure $\mathcal{K}$ with a distinguished node w we have that $\mathcal{K}, w \models \psi$ if and only if $\mathcal{K} \models \psi^*(w)$. This translation takes an atomic proposition P to the atom Px, it commutes with the Boolean connectives, and it translates the modal operators by quantifiers as follows:

$$\langle a \rangle \psi \rightsquigarrow (\langle a \rangle \psi)^*(x) := \exists y(E_a xy \wedge \psi^*(y))$$
$$[a]\psi \rightsquigarrow ([a]\psi)^*(x) := \forall y(E_a xy \rightarrow \psi^*(y))$$

where E_a is the transition relation associated with the modality a. The *modal fragment* of first-order logic is the image of propositional modal logic under this translation. Clearly the translation of modal logic into first-order logic uses only guarded quantification, so we see immediately that the modal fragment is contained in GF. In fact the guarded fragment can be viewed as the generalization of the modal fragment of FO that is obtained by dropping the restrictions to use only two variables and only monadic and binary predicates, and to retain only the restriction that quantifiers must be guarded.

The following properties of GF have been demonstrated [1,5]:

(1) The satisfiability problem for GF is decidable.
(2) GF has the finite model property, i.e., every satisfiable formula in the guarded fragment has a finite model.
(3) GF has (a generalized variant of) the tree model property.
(4) Many important model theoretic properties which hold for first-order logic and modal logic, but not, say, for the bounded-variable fragments FO^k, do hold also for the guarded fragment.

(5) The notion of equivalence under guarded formulae can be characterized by a straightforward generalization of bisimulation.

Based on this kind of results Andréka, van Benthem and Németi put forward the 'thesis' that it is the guarded nature of quantification that is the main responsible for the good model-theoretic and algorithmic properties of modal logics.

Let us discuss to what extent this explanation is adequate. One way to address this question is to look at the complexity of GF. We have shown in [5] that the the satisfiability problems for GF is complete for 2EXPTIME, the class of problems solvable by a deterministic algorithm in time $2^{2^{p(n)}}$, for some polynomial $p(n)$. This seems very bad, in particular if we compare it to the well-known fact that the satisfiability problem for propositional modal logic is in PSPACE. But dismissing the explanation of Andréka, van Benthem and Németi on these grounds would be too superficial. Indeed, the reason for the double exponential time complexity of GF is just the fact that predicates may have unbounded arity (wheras ML only expresses properties of graphs). Given that even a single predicate of arity n over a domain of just two element leads to 2^{2^n} possible types already on the atomic level, the double exponential lower complexity bound is hardly a surprise. Further, in most of the potential applications of guarded logics the vocabulary is fixed and the arity therefore bounded. But for GF-sentences of bounded arity, the satisfiability problem can be decided in EXPTIME [5], which is a complexity level that is reached already for rather weak extensions of ML (e.g. by a universal modality). Thus, the complexity analysis does not really provide a decisive answer to our question.

To approach the question from a different angle, let us look at extensions of ML. Indeed ML is a very weak logic, and the really interesting modal logics extend ML by features like path quantification, temporal operators, least and greatest fixed points etc. which are of crucial importance for most computer science applications. It has turned out that many of these extended modal logics are algorithmically still manageable and actually of considerable practical importance. The most important of these extension is the modal μ-calculus L_μ, which extends ML by least and greeatest fixed points and subsumes most of the modal logics used for automatic verification including CTL, LTL, CTL*, PDL and also many description logics. The satisfiability problem for L_μ is known to be decidable and complete for EXPTIME [3]. Therefore, a good test for the explanation put forward by Andréka, van Benthem and Németi is the following problem:

> If we extend GF by least and greatest fixed points, do we still get a decidable logic? If yes, what is its complexity? To put it differently, what is the penalty, in terms of complexity, that we pay for adding fixed points to the guarded fragment?

In [6] we were able to give a positive answer to this question. The model-theoretic and algorithmic methods that are available for the μ-calculus on one side, and the guarded fragments of first-order logic on the other side, can indeed be combined and generalized to provide positive results for guarded fixed point logics. (Precise definitions for these logics will be given in the next section.) In fact we can establish precise complexity bounds.

Theorem 1 (Grädel, Walukiewicz). *The satisfiability problems for guarded fixed point logics are decidable and* 2EXPTIME*-complete. For guarded fixed point sentences of bounded width the satisfiability problem is* EXPTIME*-complete.*

By the width of a formula ψ, we mean the maximal number of free variables in the subformulae of ψ. For sentences that are guarded in the sense of GF, the width is bounded by the maximal arity of the relation symbols, but there are other variants of guarded logics where the width may be larger. Note that for guarded fixed point sentences of bounded width the complexity level is the same as for μ-calculus and for GF without fixed points.

The proof that we give in [6] relies on alternating two-way tree automata (on trees of unbounded branching), on a forgetful determinacy theorem for parity games, and on a notion of *tableaux* for guarded fixed point sentences. A tableau can be viewed as a tree representation of a structure. We associate with every guarded fixed point sentence ψ an alternating tree automaton $\mathcal{A}_\psi$ that accepts precisely the tableaux that represent models for ψ. This reduces the satisfiability problem for guarded fixed point logic to the emptiness problem for alternating two-way tree automata.

In this paper, we explain this result and discuss several ways to design decision procedures for guarded fixed point logics. All these techniques exploit one very important property of guarded logics, namely the (generalized variant of the) *tree model property*, which says that every satisfiable sentence of width k has a model of tree width at most $k-1$.

In Sect.2 we discuss different variants of guarded logics. In Sect. 3 we explain the notions of guarded bisimulations and of the unraveling of a structure, and we prove the tree model property of guarded logics. Based on the tree model property we will present in Sect. 4.2 a somewhat simpler decidability proof for guarded fixed point logic that replaces the automata theoretic machinery used in [6] by an interpretation argument into the monadic second-order theory of countable trees (SωS) which by Rabin's famous result [8] is known to be decidable. We then show in Sect. 4.3 that instead of using SωS, one can also reduce guarded fixed point logic to the μ-calculus with backwards modalities which has recently proved to be decidable (in EXPTIME actually) by Vardi [12]. Finally we sketch the direct automata theoretic approach used in [6] (but here without the explicit use of tableaux).

2 Guarded Logics

There are several ways to define more general guarded logics than GF. On one side, we can consider other notions of guardedness, and on the other side we can look at guarded fragments of more powerful logics than first-order logic. We first consider other guardedness conditions.

Loosely Guarded Quantification. The direct translation of temporal formulae (ψ **until** φ), say over the temporal frame $(\mathbb{N}, <)$, into first-order logic is

$$\exists y(x \leq y \land \varphi(y) \land \forall z((x \leq z \land z < y) \rightarrow \psi(z))$$

which is not guarded in the sense of Definition 1. However, the quantifier $\forall z$ in this formula is guarded in a weaker sense, which lead van Benthem [2] to the following generalization of GF.

Definition 2. The *loosely guarded fragment* LGF is defined in the same way as GF, but the quantifier-rule is relaxed as follows:

(3)′ If $\psi(\boldsymbol{x}, \boldsymbol{y})$ is in LGF, and $\alpha(\boldsymbol{x}, \boldsymbol{y}) = \alpha_1 \land \cdots \land \alpha_m$ is a conjunction of atoms, then

$$\exists \boldsymbol{y}((\alpha_1 \land \cdots \land \alpha_m) \land \psi(\boldsymbol{x}, \boldsymbol{y})) \quad \text{and} \quad \forall \boldsymbol{y}((\alpha_1 \land \cdots \land \alpha_m) \rightarrow \psi(\boldsymbol{x}, \boldsymbol{y}))$$

belong to LGF, provided that $\text{free}(\psi) \subseteq \text{free}(\alpha) = \boldsymbol{x} \cup \boldsymbol{y}$ and for any two variables $z \in \boldsymbol{y}$, $z' \in \boldsymbol{x} \cup \boldsymbol{y}$ there is at least one atom α_j that contains both z and z'.

In the translation of (ψ **until** φ) described above, the quantifier $\forall z$ is loosely guarded by $(x \leq z \land z < y)$ since z coexists with both x and y in some conjunct of the guard. On the other side, the transitivity axiom $\forall xyz(Exy \land Eyz \rightarrow Exz)$ is not in LGF. The conjunction $Exy \land Eyz$ is not a proper guard of $\forall xyz$ since x and z do not coexist in any conjunct. Indeed, it has been shown in [5] that there is no way to express transitivity in LGF.

Clique-Guarded Quantification. In this paper we introduce a new, even more liberal, variant of guarded quantification, which leads to what we may call *clique-guarded logics*. To motivate this notion, let us look at the semantic meaning of guardedness.

Definition 3. Let $\mathfrak{B}$ be a structure with universe B and vocabulary τ. A set $X \subseteq B$ is *strictly k-guarded* in $\mathfrak{B}$ if, for some $s \leq k$, there exists $b_1, \ldots, b_s \in B$ such that

(i) $X \subseteq \{b_1, \ldots, b_s\}$, and
(ii) there exists an atom $\alpha(b_1, \ldots, b_s)$, in which all of $b_1, \ldots, b_s$ occur, such that $\mathfrak{B} \models \alpha(b_1, \ldots, b_s)$.

To put it differently, X is strictly k-guarded if there is a guard α in the sense of GF that guards a set containing X with cardinality at most k.

Similarly, a set $X \subseteq B$ is *loosely k-guarded* if it is contained in a set $\{b_1,\ldots, b_s\}$ (with $s \leq k$) that is guarded in the sense of LGF. That is clause *(ii)* is replaced by *(ii)′* There exists a conjunction $\alpha(b_1, \dots, b_s)$ of atoms such that every pair b_i, b_j coexists in some conjunct of α and $\mathfrak{B} \models \alpha(b_1, \dots, b_s)$.

A set is (strictly or loosely) guarded if it is (strictly or loosely) k-guarded for some k, and a tuple $\boldsymbol{b} = (b_1, \dots, b_r)$ is (strictly or loosely) guarded in $\mathfrak{B}$ if the set $\{b_1, \dots, b_r\}$ is.

Clearly, sentences of GF resp. LGF can refer only to strictly resp. loosely guarded tuples. Note that the components of a loosely guarded tuple need not coexist in a single atom, but they are nevertheless all 'adjacent' in the sense of the locality graph or Gaifman graph of a structure, which is an important notion in finite model theory. Let us recall the definition.

Definition 4. The *Gaifman graph* of a relational structure $\mathfrak{B}$ (with universe B) is the undirected graph $G(\mathfrak{B}) = (B, E^{\mathfrak{B}})$ where

$$E^{\mathfrak{B}} = \{(a, a') : a \neq a', \text{there exists a guarded set } X \subseteq B \text{ with } a, a' \in X\}.$$

The following observation is obvious.

Lemma 1. *Let X be loosely guarded in $\mathfrak{B}$. Then X induces a clique in $G(\mathfrak{B})$.*

The converse is not true, as the following example shows. Consider a structure $\mathfrak{A} = (A, R)$ with universe $A = \{a_1, a_2, a_3, b_{12}, b_{23}, b_{13}\}$ and one ternary relation R containing the triangles $(a_1, a_1, b_{12}), (a_2, a_3, b_{23})(a_1, a_3, b_{13})$. Then the tuple (a_1, a_2, a_3) is neither guarded nor loosely guarded, but induces a clique in $G(\mathfrak{A})$.

Definition 5. For each finite vocabulary τ and each $k \in \mathbb{N}$, there is a positive, existential first-order formula $\mathit{clique}(x_1, \dots, x_k)$ such that, for every τ-structure $\mathfrak{B}$ and all $b_1, \dots, b_k \in B$

$$\mathfrak{B} \models \mathit{clique}(b_1, \dots, b_k) \iff b_1, \dots, b_k \text{ induce a clique in } G(\mathfrak{B}).$$

A tuple $b_1, \dots, b_k$ such that $\mathfrak{B} \models \mathit{clique}(b_1, \dots, b_k)$ is called *clique-guarded* in $\mathfrak{B}$.

Definition 6. The *clique-guarded fragment* CGF of first-order logic is defined in the same way as GF and LGF, but with the *clique*-formulae as guards. Hence, the quantification rule for CGF is

(3)″ If $\psi(\boldsymbol{x}, \boldsymbol{y})$ is a formula CGF, then

$$\exists \boldsymbol{y}(\mathit{clique}(\boldsymbol{x}, \boldsymbol{y}) \land \psi(\boldsymbol{x}, \boldsymbol{y})) \quad \text{and} \quad \forall \boldsymbol{y}(\mathit{clique}(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \psi(\boldsymbol{x}, \boldsymbol{y}))$$

belong to CGF, provided that $\text{free}(\psi) \subseteq \text{free}(\mathit{clique}) = \boldsymbol{x} \cup \boldsymbol{y}$.

We observe that CGF has strictly more expressive power than LGF.

Proposition 1. *The* CGF*-sentence $\forall xyz(\mathit{clique}(x, y, z) \rightarrow Rxyz)$ is not equivalent to any sentence in* LGF*.*

On the other side CGF inherits all the nice decidability and model-theoretic properties of LGF.

Notation. We use the notation $(\exists \boldsymbol{y} \,.\, \alpha)$ and $(\forall \boldsymbol{y} \,.\, \alpha)$ for relativized quantifiers, i.e., we write guarded formulae in the form $(\exists \boldsymbol{y} \,.\, \alpha)\psi(\boldsymbol{x}, \boldsymbol{y})$ and $(\forall \boldsymbol{y} \,.\, \alpha)\psi(\boldsymbol{x}, \boldsymbol{y})$. When this notation is used, then it is always understood that α is indeed a proper guard as specified by condition (3), (3)$'$ or (3)$''$.

Remark. Note that quantifiers over tuples are in principle no longer needed in CGF (contrary to GF and LGF), since they can be written as sequences of clique-guarded quantifiers over single variables.

Guarded Fixed Point Logics. We now define guarded fixed point logics, which can be seen as the natural common extensions of GF, LGF and CGF on one side, and the μ-calculus on the other side.

Definition 7. The guarded fixed point logics μGF, μLGF and μCGF are obtained by adding to GF, LGF and CGF, respectively, the following rules for constructing fixed-point formulae:

Let W be a k-ary relation symbol, $\boldsymbol{x} = x_1, \dots, x_k$ a k-tuple of distinct variables and $\psi(W, \boldsymbol{x})$ be a guarded formula that contains only positive occurrences of W, no free first-order variables other than $x_1, \dots, x_k$ and where W is not used in guards. Then we can build the formulae

$$[\text{LFP } W\boldsymbol{x} \,.\, \psi](\boldsymbol{x}) \qquad \text{and} \qquad [\text{GFP } W\boldsymbol{x} \,.\, \psi](\boldsymbol{x}).$$

The semantics of the fixed point formulae is the usual one: Given a structure $\mathfrak{A}$ providing interpretations for all free second-order variables in ψ, except W, the formula $\psi(W, \boldsymbol{x})$ defines an operator on k-ary relations $W \subseteq A^k$, namely

$$\psi^{\mathfrak{A}} : W \mapsto \psi^{\mathfrak{A}}(W) := \{\boldsymbol{a} \in A^k : \mathfrak{A} \models \psi(W, \boldsymbol{a})\}.$$

Since W occurs only positively in ψ, this operator is monotone (i.e., $W \subseteq W'$ implies $\psi^{\mathfrak{A}}(W) \subseteq \psi^{\mathfrak{A}}(W')$) and therefore has a least fixed point $\text{LFP}(\psi^{\mathfrak{A}})$ and a greatest fixed point $\text{GFP}(\psi^{\mathfrak{A}})$. Now, the semantics of least fixed point formulae is defined by

$$\mathfrak{A} \models [\text{LFP } W\boldsymbol{x} \,.\, \psi(W, \boldsymbol{x})](\boldsymbol{a}) \quad \text{iff} \quad \boldsymbol{a} \in \text{LFP}(\psi^{\mathfrak{A}})$$

and similarly for the greatest fixed points.

The Löwenheim-Skolem Property. For future use, we recall that even the (unguarded) least fixed point logic (FO + LFP) has the Löwenheim-Skolem property. This result is part of the folklore on fixed point logic [4].

Theorem 2. *Every satisfiable sentence in* (FO + LFP)*, and hence every satisfiable sentence in* μGF*,* μLGF *or* μCGF*, has a model of countable cardinality.*

On the other side, guarded fixed point logics do *not* have the *finite model property* (see [6] for a simple counterexample).

Guarded Infinitary Logics. It is well known that fixed point logics have a close relationship to infinitary logics (with bounded number of variables).

Definition 8. $\mathrm{GF}^\infty, \mathrm{LGF}^\infty$ and CGF^∞ are the infinitary variants of the guarded fragments GF, LGF and CGF, respectively. For instance GF^∞ extends GF by the following rule for building new formulae: If $\Phi \subseteq \mathrm{GF}^\infty$ is any set of formulae, then also $\bigvee \Phi$ and $\bigwedge \Phi$ are formulae of GF^∞. The definitions for LGF^∞ and CGF^∞ are anologous.

In the sequel we explicitly talk about the clique-guarded case only, i.e. about μCGF and CGF^∞, but all results apply to the guarded and loosely guarded case as well. The following simple observation relates μCGF with CGF^∞.

Proposition 2. *For each $\psi \in \mu\mathrm{CGF}$ of width k and each cardinal γ, there is a $\psi' \in \mathrm{CGF}^\infty$, also of width k, which is equivalent to ψ on all structures up to cardinality γ.*

3 Guarded Bisimulation and the Tree Model Property

Tree width is an important notion in graph theory. Here we need a generalisation of this concept to arbitrary relational structures. For readers who are familiar with the notion of tree width in graph theory we can simply say that the tree width of a structure is the tree width of its Gaifman graph. Here is a more detailed definition.

Definition 9. A structure $\mathfrak{B}$ (with universe B and arbitrary vocabulary τ) has *tree width k* if k is the minimal natural number satisfying the following condition. There exists a directed tree $T = (V, E)$ and a function

$$F : V \to \{X \subseteq B : |X| \leq k+1\},$$

assigning to every node v of T a set $F(v)$ of at most $k+1$ elements of $\mathfrak{B}$, such that the following two conditions hold.

(i) For every τ-atom $\alpha(x_1, \dots, x_r)$ and every tuple $b_1, \dots, b_r$ such that $\mathfrak{B} \models \alpha(b_1, \dots, b_r)$ there exists a node v of T such that $\{b_1, \dots, b_r\} \subseteq F(v)$.
(ii) For every element b of $\mathfrak{B}$, the set of nodes $\{v \in V : b \in F(v)\}$ is connected (and hence induces a subtree of T).

For each node v of T, the set $F(v)$ induces a substructure $\mathfrak{F}(v) \subseteq \mathfrak{B}$ of cardinality at most $k+1$. (Since $F(v)$ may be empty, we also permit empty substructures.) $\langle T, (\mathfrak{F}(v)_{v\in T})\rangle$ is called a *tree decomposition* of width k of $\mathfrak{B}$.

Remark. A more concise, but equivalent, formulation of clause *(i)* would be that $\mathfrak{B} = \bigcup_{v\in T} \mathfrak{F}(v)$.

By definition, every strictly guarded set $X \subseteq B$ is contained in some $F(v)$. A simple graph theoretic argument shows that the same is true for loosely guarded and clique-guarded sets.

Lemma 2. *Let $\langle T, (\mathfrak{F}(v)_{v\in T})\rangle$ be a tree decomposition of $\mathfrak{B}$ and $X \subseteq B$ be a clique-guarded set in $\mathfrak{B}$. Then there exists a node v of T such that $X \subseteq F(v)$.*

Proof. For each $b \in X$, let V_b be the set of nodes v such that $b \in F(v)$. By the definition of a tree decomposition, each V_b induces a subtree of T. For all $b, b' \in X$ the intersection $V_b \cap V_{b'}$ is non-empty, since b and b' are adjacent in $G(\mathfrak{B})$ and must therefore coexist in some atomic fact that is true in $\mathfrak{B}$. It is known that any collection of pairwise overlapping subtrees of a tree has a common node (see e.g. [9, p. 94]). Hence there is a node v of the T such that $F(v)$ contains all elements of X. □

Guarded Bisimulations. The notion of bisimulation from modal logic generalises in a straightforward way to various notions of guarded bisimulation that describe indistinguishability in guarded logics. We focus here on *clique-k-bisimulations*, the appropriate notion for clique-guarded formulae of width at most k. The notions of strict or loose k-bisimulations can be defined analogously.

Definition 10. A *clique-k-bisimulation* (in the sequel simply called a k-bisimulation), between two τ-structures $\mathfrak{A}$ and $\mathfrak{B}$ is a non-empty set I of finite partial isomorphisms $f : X \to Y$ from $\mathfrak{A}$ to $\mathfrak{B}$, where $X \subseteq A$ and $Y \subseteq B$ are clique-guarded sets of size at most k, such that the following back and forth conditions are satisfied. For every $f : X \to Y$ in I,

forth: for every clique-guarded set $X' \subseteq A$ of size at most k there exists a partial isomorphism $g : X' \to Y'$ in I such that f and g agree on $X \cap X'$.
back: for every clique-guarded set $Y' \subseteq B$ of size at most k there exists a partial isomorphism $g : X' \mapsto Y'$ in I such that f^{-1} and g^{-1} agree on $Y \cap Y'$.

Two τ-structures $\mathfrak{A}$ and $\mathfrak{B}$ are *(clique-)k-bisimilar* if there exists a k-bisimulation between them.

Remark. One can describe k-bisimilarity also via a guarded variant of the infinitary Ehrenfeucht-Fraïssé game with k pebbles. One just has to impose that after every move, the set of all pebbled elements induces a clique in the Gaifman graph of each of the two structures. Then $\mathfrak{A}$ and $\mathfrak{B}$ are k-bisimilar if and only if Player II has a winning strategy for this guarded game.

Adapting basic and well-known model-theoretic techniques to the present situation, one obtains the following result.

Theorem 3. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be two τ-structures. The following are equivalent:*

(i) $\mathfrak{A}$ and $\mathfrak{B}$ are k-bisimilar.
(ii) For all sentences $\psi \in \mathrm{CGF}^\infty$ of width at most k, $\mathfrak{A} \models \psi \iff \mathfrak{B} \models \psi$.

In particular this shows that k-bisimilar structures cannot be separated by guarded fixed point sentences of width k.

Unravelings of Structures. The k-unraveling $\mathfrak{B}^{(k)}$ of a structure $\mathfrak{B}$ is defined inductively. We build a tree T, with functions F and G such that each

$F(v)$ induces a clique-guarded substructure $\mathfrak{F}(v) \subseteq \mathfrak{B}$, each $G(v)$ induces a substructure $\mathfrak{G}(v) \subseteq \mathfrak{B}^{(k)}$ that is isomorphic to $\mathfrak{F}(v)$, and $\langle T, (\mathfrak{G}(v))_{v \in T}\rangle$ is a tree decomposition of $\mathfrak{B}^{(k)}$.

The root of T is λ, with $F(\lambda) = G(\lambda) = \varnothing$. Given a node v of T with $F(v) = \{a_1, \dots, a_r\}$ and $G(v) = \{a_1^*, \dots, a_r^*\}$ we create for every clique-guarded set $\{b_1, \dots, b_s\}$ in $\mathfrak{B}$ with $s \leq k$ a successor node w of v such that $F(w) = \{b_1, \dots, b_s\}$ and $G(w)$ is a set $\{b_1^*, \dots, b_s^*\}$ which is defined as follows. For those i, such that $b_i = a_j \in F(v)$, put $b_i^* = a_j^*$ so that $G(w)$ has the same overlap with $G(v)$ as $F(w)$ has with $F(v)$. The other b_i^* in $G(w)$ are fresh elements.

Let $f_w : F(w) \to G(w)$ be the bijection taking b_i to b_i^* for $i = 1, \dots, s$. For $\mathfrak{F}(w)$ being the substructure of $\mathfrak{B}$ induced by $F(w)$, define $\mathfrak{G}(w)$ so that f_w is an isomorphism from $\mathfrak{F}(w)$ to $\mathfrak{G}(w)$. Finally $\mathfrak{B}^{(k)}$ is the structure with tree decomposition $\langle T, (\mathfrak{G}(v)_{v \in T})\rangle$.

Note that the k-unraveling of a structure has tree width at most $k-1$.

Proposition 3. *$\mathfrak{B}$ and $\mathfrak{B}^{(k)}$ are k-bisimilar.*

Proof. Let I be the set of functions $f_v : F(v) \to G(v)$ for all nodes v of T. $\square$

It follows that no sentence of width k in CGF$^\infty$, and hence no sentence of width k in μCGF distinguishes between a structure and its k-unraveling. Since every satisfiable sentence in μCGF has a model of at most countable cardinality, and since the k-unraveling of a countable model is again countable we obtain the following tree model property for guarded fixed point logic.

Theorem 4 (Tree Model Property). *Every satisfiable sentence ψ in μCGF of width k has a countable model of tree width at most $k-1$.*

Remark. In fact our decision algorithms will imply a stronger version of the tree model property, where the underlying tree has branching bounded by $O(|\psi|^k)$.

4 Decision Procedures

Once the tree model property is established, there are several ways to design decision algorithms for guarded logics. We focus here on guarded fixed point logics (in fact on μCGF which contains μGF and μLGF).

4.1 Tree Representations of Structures

Let $\langle T, (\mathfrak{F}(v))_{v \in T}\rangle$ be a tree decomposition of width $k-1$ of a τ-structure $\mathfrak{D}$ with universe D. We want to describe $\mathfrak{D}$ by a tree with a finite set of labels. To this end, we fix a set K of $2k$ constants and choose a function $f : D \to K$ assigning to each element d of $\mathfrak{D}$ a constant $a_d \in K$ such that the following condition is satisfied. If v, w are adjacent nodes of T, then distinct elements of $\mathfrak{F}(v) \cup \mathfrak{F}(w)$ are always mapped to distinct constants of K.

For each constant $a \in K$, let $\mathcal{O}_a$ be the set of those nodes $v \in T$ at which the constant a occurs, i.e., for which there exists an element $d \in \mathfrak{F}(v)$ such that $f(d) = a$. Further, we introduce for each m-ary relation R of $\mathfrak{D}$ a tuple $\overline{R} := (R_{\boldsymbol{a}})_{\boldsymbol{a} \in K^m}$ of monadic relations on T with

$$\begin{aligned} R_{\boldsymbol{a}} := \{v \in T : \ & \text{there exist } d_1, \dots, d_m \in \mathfrak{F}(v) \text{ such that} \\ & \mathfrak{F}(v) \models Rd_1 \cdots d_m \text{ and } f(d_1) = a_1, \dots, f(d_m) = a_m\}. \end{aligned}$$

The tree $T = (V, E)$ together with the monadic relations $\mathcal{O}_a$ and $R_{\boldsymbol{a}}$ (for $R \in \tau$) is called the tree structure $\mathcal{T}(\mathfrak{D})$ associated with $\mathfrak{D}$ (and, strictly speaking, with its tree decomposition and with K and f). Note that two occurrences of a constant $a \in K$ at nodes u, v of T represent the same element of $\mathfrak{D}$ if and only if a occurs in the label of *all* nodes on the link between u and v. (The link between two nodes u, v in a tree T is the smallest connected subgraph of T containing both u and v.)

An arbitrary tree $T = (V, E)$ with monadic relations $\mathcal{O}_a$ and $\overline{R}$ does define a tree decomposition of width $k - 1$ of some structure $\mathfrak{D}$, provided that the following axioms are satisfied.

(1) At each node v, at most k of the predicates $\mathcal{O}_a$ are true.
(2) Neighbouring nodes agree on their common elements. For all m-ary relation symbols $R \in \tau$ we have the axiom

$$\mathit{consistent}(\overline{R}) := \bigwedge_{\boldsymbol{a} \in K^m} \forall x \forall y \Big(\Big(Exy \wedge \bigwedge_{a \in \boldsymbol{a}} (\mathcal{O}_a x \wedge \mathcal{O}_a y) \Big) \rightarrow (R_{\boldsymbol{a}} x \leftrightarrow R_{\boldsymbol{a}} y) \Big).$$

These are first-order axioms over the vocabulary $\tau^* := \{E\} \cup \{\mathcal{O}_a : a \in K\} \cup \{R_{\boldsymbol{a}} : R \in \tau, \boldsymbol{a} \subseteq K\}$. Given a tree structure $\mathcal{T}$ with underlying tree $T = (V, E)$ and monadic predicates $\mathcal{O}_a$ and $R_{\boldsymbol{a}}$ satisfying (1) and (2), we obtain a structure $\mathfrak{D}$ such that $\mathcal{T}(\mathfrak{D}) = \mathcal{T}$ as follows. For every constant $a \in K$, we call two nodes u, w of T a-equivalent if $\mathcal{T} \models \mathcal{O}_a v$ for all nodes v on the link between u and w. Clearly this is an equivalence relation on $\mathcal{O}_a^{\mathcal{T}}$. We write $[v]_a$ for the a-equivalence class of the node v. The universe of $\mathfrak{D}$ is the set of all a-equivalence classes of T for $a \in K$, i.e.,

$$D := \{[v]_a : v \in T,\ a \in K,\ \mathcal{T} \models \mathcal{O}_a v\}.$$

For every m-ary relation symbol R in τ, we define

$$\begin{aligned} R^{\mathfrak{D}} := \{([v_1]_{a_1}, \dots, [v_m]_{a_m}) : & \mathcal{T} \models R_{a_1 \dots a_m} v \text{ for some} \\ & \text{(and hence all) } v \in [v_1]_{a_1} \cap \cdots \cap [v_m]_{a_m}\}. \end{aligned}$$

4.2 Reduction to SωS

We now describe a translation from μCGF into monadic second-order logic on countable trees. Given a formula $\varphi(x_1, \dots, x_m) \in \mu$CGF and a tuple $\boldsymbol{a} =$

$a_1, \dots, a_m$ over K, we construct a monadic second-order formula $\varphi_{\boldsymbol{a}}(z)$ with one free variable. The formulae $\varphi_{\boldsymbol{a}}(z)$ describe in the associated tree structure $\mathcal{T}(\mathfrak{D})$ the same properties of clique-guarded tuples as $\varphi(\boldsymbol{x})$ does in $\mathfrak{D}$. (We will make this statement more precise below).

On a directed tree $T = (V, E)$ we can express that U contains all nodes on the link between x and y by the formula

$$\begin{aligned} \mathit{connect}(U, x, y) := Ux \wedge Uy \wedge \exists r(Ur \wedge \forall z(Ezr \rightarrow \neg Uz) \\ \wedge \forall w \forall z(Ewz \wedge Uz \wedge z \neq r \rightarrow Uw)). \end{aligned}$$

For any set $\boldsymbol{a} \subseteq K$ we can then construct a monadic second-order formula

$$\mathit{link}_{\boldsymbol{a}}(x, y) := \exists U\Big(\mathit{connect}(U, x, y) \wedge \forall z(Uz \rightarrow \bigwedge_{a \in \boldsymbol{a}} O_a z)\Big)$$

saying that the tuple $\boldsymbol{a}$ occurs at all nodes on the link between x and y. The translation is now defined by induction as follows:

(1) If $\varphi(\boldsymbol{x})$ is an atom $Sx_{i_1} \cdots x_{i_m}$ then $\varphi_{\boldsymbol{a}}(z) := S_{\boldsymbol{b}} z$ where $\boldsymbol{b} = (a_{i_1}, \dots, a_{i_m})$.
(2) If $\varphi = (x_i = x_j)$, let $\varphi_{\boldsymbol{a}}(z) = \mathit{true}$ if $a_i = a_j$ and $\varphi_{\boldsymbol{a}}(z) = \mathit{false}$ otherwise.
(3) If $\varphi(\boldsymbol{x}) := \mathit{clique}(\boldsymbol{x})$, let

$$\mathit{clique}_{\boldsymbol{a}}(z) := \bigwedge_{a, a \in \boldsymbol{a}} \exists y(\mathit{link}_{a,a'}(y, z) \wedge \bigvee_{R \in \tau} \bigvee_{\boldsymbol{b}: a, a' \in \boldsymbol{b}} R_{\boldsymbol{b}} y).$$

(4) If $\varphi = \eta \wedge \vartheta$, let $\varphi_{\boldsymbol{a}}(z) = \eta_{\boldsymbol{a}}(z) \wedge \vartheta_{\boldsymbol{a}}(z)$.
(5) If $\varphi = \neg\vartheta$, let $\varphi_{\boldsymbol{a}}(z) = \neg\vartheta_{\boldsymbol{a}}(z)$.
(6) If $\varphi = (\exists \boldsymbol{y} \,.\, \mathit{clique}(\boldsymbol{x}, \boldsymbol{y}))\eta(\boldsymbol{x}, \boldsymbol{y})$, let

$$\varphi_{\boldsymbol{a}}(z) := \exists y\Big(\mathit{link}_{\boldsymbol{a}}(y, z) \wedge \bigvee_{\boldsymbol{b}} \Big(\bigwedge_{b \in \boldsymbol{b}} O_b y \wedge \mathit{clique}_{\boldsymbol{ab}}(y) \wedge \eta_{\boldsymbol{ab}}(y)\Big)\Big).$$

(7) If $\varphi = [\mathrm{LFP} \;\; S\boldsymbol{x} \,.\, \eta(S, \boldsymbol{x})](\boldsymbol{x})$, let

$$\varphi_{\boldsymbol{a}}(z) := \forall \overline{S}\Big(\Big(\mathit{consistent}(\overline{S}) \wedge \bigwedge_{\boldsymbol{b}} \forall x(S_{\boldsymbol{b}} x \leftrightarrow \eta_{\boldsymbol{b}}(\overline{S}, x))\Big) \rightarrow S_{\boldsymbol{a}} z\Big).$$

Here $\overline{S}$ is a tuple $(S_{\boldsymbol{b}})_{\boldsymbol{b} \in K^m}$ of monadic predicates where m is the arity of S.

Theorem 5. *Let $\varphi(\boldsymbol{x})$ be a formula in μCGF and $\mathfrak{D}$ be a structure with tree decomposition $\langle T, (\mathfrak{F}(v))_{v \in T} \rangle$. For an appropriate set of constants K and a function $f : D \rightarrow K$, let $\mathcal{T}(\mathfrak{D})$ be the associated tree structure. Then, for every node v of T and every clique-guarded tuple $\boldsymbol{d} \subseteq \mathfrak{F}(v)$ with $f(\boldsymbol{d}) = \boldsymbol{a}$,*

$$\mathfrak{D} \models \varphi(\boldsymbol{d}) \iff \mathcal{T}(\mathfrak{D}) \models \varphi_{\boldsymbol{a}}(v).$$

Proof. We proceed by induction on φ. The only non-trivial cases are the clique-guards, existential quantification and least fixed points.

For the clique-guards, note that the translated formula $\mathit{clique}_{\boldsymbol{a}}(v)$ says that for any pair a, a' of components of $\boldsymbol{a}$, there is a node w, such that

– a, a' occur at all nodes on the link from v to w and hence represent the same elements d, d' at w as they do at v.
– $T(\mathfrak{D}) \models R_{\boldsymbol{b}}w$ for some predicate R and some tuple $\boldsymbol{b}$ that contains both a and a'. By induction hypothesis, this means that d, d' are components of some tuple $\boldsymbol{d'}$ such that $\mathfrak{D} \models R\boldsymbol{d'}$.

Hence $\mathcal{T}(\mathfrak{D}) \models \mathit{clique}_{\boldsymbol{a}}(v)$ if and only if the tuple $\boldsymbol{d}$ induces a clique in in the Gaifman graph $G(\mathfrak{D})$.

Suppose now that $\varphi(\boldsymbol{x}) = (\exists \boldsymbol{y} \,.\, \mathit{clique}(\boldsymbol{x}, \boldsymbol{y}))\eta(\boldsymbol{x}, \boldsymbol{y})$ and that $\mathfrak{D} \models \varphi(\boldsymbol{d})$. Then there exists a tuple $\boldsymbol{d'}$ such that $\mathfrak{D} \models \mathit{clique}(\boldsymbol{d}, \boldsymbol{d'}) \wedge \eta(\boldsymbol{d}, \boldsymbol{d'})$.

By Lemma 2 there exists a node w of T such that all components of $\boldsymbol{d} \cup \boldsymbol{d'}$ are contained in $\mathfrak{F}(w)$.

Let $f(\boldsymbol{d'}) = \boldsymbol{b}$. By induction hypothesis it follows that

$$\mathcal{T}(\mathfrak{D}) \models \bigwedge_{b \in \boldsymbol{b}} \mathcal{O}_b w \wedge \mathit{clique}_{\boldsymbol{ab}}(w) \wedge \eta_{\boldsymbol{ab}}(w).$$

Let U be the set of nodes on the link between v and w. Then the tuple $\boldsymbol{d}$ occurs in $\mathfrak{F}(u)$ for all nodes $u \in U$. It follows that $\mathcal{T}(\mathfrak{D}) \models \mathit{link}_{\boldsymbol{a}}(v, w)$. Hence $\mathcal{T}(\mathfrak{D}) \models \varphi_{\boldsymbol{a}}(v)$.

Conversely, if $\mathcal{T}(\mathfrak{D}) \models \varphi_{\boldsymbol{a}}(v)$ then there exists a node w such that the constants $\boldsymbol{a}$ occur at all nodes on the link between v and w (and hence correspond to the same tuple $\boldsymbol{d}$) and such that $\mathcal{T}(\mathfrak{D}) \models \mathit{clique}_{\boldsymbol{ab}}(w) \wedge \eta_{\boldsymbol{ab}}(w)$ for some tuple $\boldsymbol{b}$. By induction hypothesis this implies that $\mathfrak{D} \models \mathit{clique}(\boldsymbol{d}, \boldsymbol{d'}) \wedge \eta(\boldsymbol{d}, \boldsymbol{d'})$ for some tuple $\boldsymbol{d'}$, hence $\mathfrak{D} \models \varphi(\boldsymbol{d})$.

Finally, let $\varphi(\boldsymbol{x}) = [\mathrm{LFP}\ \ S\boldsymbol{x} \,.\, \eta(S, \boldsymbol{x})](\boldsymbol{x})$. By definition, $\mathfrak{D} \models \varphi(\boldsymbol{d})$ is true if and only if $\boldsymbol{d}$ is contained in every fixed point of the operator $\eta^{\mathfrak{D}}$, i.e. is in every relation S such that $S = \{\boldsymbol{c} : (\mathfrak{D}, S) \models \eta(S, \boldsymbol{c})\}$.

We first observe that, for guarded tuples $\boldsymbol{d}$, this is equivalent to the seemingly weaker condition that $\boldsymbol{d}$ is contained in every S such that $\boldsymbol{c} \in S$ iff $\mathfrak{D} \models \eta(S, \boldsymbol{c})$ *for all guarded tuples* $\boldsymbol{c}$. Indeed this is obvious, since $\eta(S, \boldsymbol{x})$ is a Boolean combination of quantifier-free formulae not involving $\boldsymbol{x}$, of positive atoms of the form $S\boldsymbol{u}$ where $\boldsymbol{u}$ is a recombination of the variables appearing in $\boldsymbol{x}$ and of formulae starting with a guarded existential quantifier. Therefore the truth values of $S\boldsymbol{c}$ for unguarded tuples $\boldsymbol{c}$ never matters for the question whether a given guarded tuple is in $\varphi^{\mathfrak{D}}(S)$.

Recall that the formula associated with $\varphi(\boldsymbol{x})$ and $\boldsymbol{a}$ is

$$\varphi_{\boldsymbol{a}}(z) := (\forall \overline{S})\Big(\Big(\mathit{consistent}(\overline{S}) \wedge \bigwedge_{\boldsymbol{b}} \forall x(S_{\boldsymbol{b}}x \leftrightarrow \eta_{\boldsymbol{b}}(\overline{S}, x))\Big) \rightarrow S_{\boldsymbol{a}}z\Big).$$

Consider any tuple $\overline{S} = (S_{\boldsymbol{b}})_{\boldsymbol{b}\in K^m}$ of monadic relations on $\mathcal{T}(\mathfrak{D})$ that satisifies the consistency axiom such that

$$(\mathcal{T}(\mathfrak{D}), \overline{S}) \models \bigwedge_{\boldsymbol{b}} \forall x(S_{\boldsymbol{b}}x \leftrightarrow \eta_{\boldsymbol{b}}(\overline{S}, x)).$$

This tuple $\overline{S}$ defines a relation S on $\mathfrak{D}$ such that for all nodes w of T and all tuples $\boldsymbol{c}$ in $\mathfrak{F}(w)$ with $f(\boldsymbol{c}) = \boldsymbol{b}$, we have $\boldsymbol{c} \in S$ iff $w \in S_{\boldsymbol{b}}$. Conversely each relation S on $\mathfrak{D}$ defines such a tuple $\overline{S}$ of monadic relations on $\mathcal{T}(\mathfrak{D})$ which describes the truth values of S on all guarded tuples of $\mathfrak{D}$. Since $\mathcal{T}(\mathfrak{D}) \models S_{\boldsymbol{b}}w \leftrightarrow \eta_{\boldsymbol{b}}(\overline{S}, w)$ it follows by induction hypothesis that $\mathfrak{D} \models S\boldsymbol{c} \leftrightarrow \eta(S, \boldsymbol{c})$. Further $\boldsymbol{d} \in S$ if and only if $v \in S_{\boldsymbol{a}}$.

Hence the formula $\varphi_{\boldsymbol{a}}(v)$ is true in $\mathcal{T}(\mathfrak{D})$ if and only if $\boldsymbol{d}$ is contained in all relations S over $\mathfrak{D}$ such that for all guarded tuples $\boldsymbol{c}$, $\boldsymbol{c} \in S$ iff $\boldsymbol{c} \in \eta^{\mathfrak{D}}(S)$. By the remarks above this is equivalent to saying that $\boldsymbol{d}$ is in the least fixed point of $\eta^{\mathfrak{D}}$. □

Theorem 6. *The satisfiability problem for* μCGF *is decidable.*

Proof. Let ψ be a sentence in μCGF of vocabulary τ and width k. We translate ψ into a monadic second-order sentence ψ^* such that ψ is satisfiable if and only if there exists a countable tree $T = (V, E)$ with $T \models \psi^*$.

Fix a set K of $2k$ constants and let $\overline{\mathcal{O}}$ be the tuple of monadic relations $\mathcal{O}_a$ for $a \in K$. Further, for each m-relation symbol $R \in \tau$, let $\overline{R}$ be the tuple of monadic relation $R_{\boldsymbol{a}}$ where $\boldsymbol{a} \in K^m$. The desired monadic second-order sentence has the form

$$\psi^* := (\exists\overline{\mathcal{O}})(\exists\overline{R})(\chi \wedge \forall x \psi_{\varnothing}(x)).$$

Here χ is the first-order axiom expressing that the tree T expanded by the relations $\overline{\mathcal{O}}$ and $\overline{R}$ does describe a tree structure $\mathcal{T}(\mathfrak{D})$ associated to some τ-structure $\mathfrak{D}$. We have shown above that this can be done in first-order logic. The formula $\psi_{\varnothing}(x)$ is the translation of ψ (and the empty tuple of constants) into monadic second-order logic, as described by Theorem 5.

If ψ is satisfiable, then by Theorem 4, ψ has a countable model $\mathfrak{D}$ of tree width $k-1$. By Theorem 5, the associated tree structure $\mathcal{T}(\mathfrak{D})$ satisfies $\chi \wedge \forall x \psi_{\varnothing}(x)$, hence there exists a tree T such that $T \models \psi^*$. Conversely, if $T \models \psi^*$, then there exists an expansion $\mathcal{T} = (T, \overline{\mathcal{O}}, \overline{R})$ which satisfies χ and hence describes the tree decomposition of a τ-structure $\mathfrak{D}$. Since $\mathcal{T} \models \forall x \psi_{\varnothing}(x)$ it follows by Theorem 5 that $\mathfrak{D} \models \psi$.

The decidability of μCGF now follows by the decidability of SωS, the monadic second-order theory of countable trees, a famous result that has been established by Rabin [8]. □

Note that while this reduction argument to SωS gives a somewhat more elementary decidability proof (modulo Rabin's result, of course), it does not give good complexity bounds. Indeed, even the first-order theory of countable trees is non-elementary, i.e. its time complexity exceeds every bounded number of iterations of the exponential function.

4.3 Reduction to the μ-Calculus with Backwards Modalities

Instead of reducing the satisfiability problem for μCGF to the monadic second-order theory of trees, we can define a similar reduction to the μ-calculus with backward modalities and then invoke Vardi's decidability result for this logic [12].

For a set of actions A, the μ-calculus with backwards modalities $L_\mu^{\leftarrow}$, permits, for each action $a \in A$, besides the common modal operators $\langle a \rangle$ and $[a]$ also the backwards operators $\langle a^{\leftarrow} \rangle$ and $[a^{\leftarrow}]$ corresponding to the backwards transitions $E_a^{\leftarrow} := \{(w,v) : (v,w) \in E_a\}$. Hence $\langle a^{\leftarrow} \rangle \varphi$ is true at state w in a Kripke structure $\mathfrak{K}$ if and only if there exists a state v, such that $\mathfrak{K}, v \models \varphi$ and w is reachable from v via action a.

Here, we will need $L_\mu^{\leftarrow}$ on trees (V,E) with only one transition relation. We can write $\langle + \rangle$, $[+]$ for the forward modal operators, and $\langle - \rangle$, $[-]$ for the backwards operators, and then use the abbreviations

$$\Diamond\varphi := \langle + \rangle\varphi \vee \langle - \rangle\varphi \quad \text{and} \quad \Box\varphi := [+]\varphi \wedge [-]\varphi.$$

Hence $\Diamond$ and $\Box$ are the usual modal operators on symmetric Kripke structures.

Finally, it is convenient for our reduction argument to permit the use of simultaneous least and greatest fixed points in $L_\mu^{\leftarrow}$. Let $\overline{X} = (X_1, \ldots, X_r)$ be a sequence of propositional variables, and $\overline{\varphi}(\overline{X}) = (\varphi_1(\overline{X}), \ldots, \varphi_r(\overline{X}))$ be a sequence of $L_\mu^{\leftarrow}$-formulae in which all occurrences of $X_1, \ldots, X_r$ are positive. Then, for each $i \le r$, the expressions $[\mu\overline{X} \,.\, \overline{\varphi}(\overline{X})]_i$ and $[\nu\overline{X} \,.\, \overline{\varphi}(\overline{X})]_i$ are formulae in $L_\mu^{\leftarrow}$.

On every Kripke structure $\mathfrak{K}$ with universe V, the sequence $\overline{\varphi}(\overline{X})$ defines an operator $\overline{\varphi}^{\mathfrak{K}}$ that maps any tuple $\overline{S} = (S_1, \ldots, S_r)$ of subsets $S_i \subseteq V$ to a new tuple $(\varphi_1^{\mathfrak{K}}(\overline{S}), \ldots, \varphi_r^{\mathfrak{K}}(\overline{S}))$ where $\varphi_i^{\mathfrak{K}}(\overline{S}) = \{v : \mathfrak{K}, v \models \varphi_i(\overline{S})\}$.

Since the variables in $\overline{X}$ occur only positive in $\overline{\varphi}$, the operator $\overline{\varphi}^{\mathfrak{K}}$ has a least fixed point $\overline{X}^{\infty} = (X_1^{\infty}, \ldots, X_r^{\infty})$. Now, the semantics of simulteneous least fixed point formulae is given by

$$\mathfrak{K}, v \models [\mu\overline{X} \,.\, \overline{\varphi}(\overline{X})]_i \iff v \in X_i^{\infty}.$$

The meaning of a simultaneous greatest fixed point $[\nu\overline{X} \,.\, \overline{\varphi}(\overline{X})]_i$ is defined similarly. It is well-known that simultaneous fixed points can be rewritten as nestings of simple fixed points, so the use of simulateneous fixed point does not change the expressive power of $L_\mu^{\leftarrow}$.

Theorem 7 (Vardi). *Every satisfiable formula in $L_\mu^{\leftarrow}$ has a tree model. Further, the satisfiability problem for $L_\mu^{\leftarrow}$ is decidable and* Exptime*-complete.*

On connected Kripke structures (in particular on trees), the universal modality is definable in $L_\mu^{\leftarrow}$. For every formula φ, we write $\forall\varphi$ to abbreviate the formula $\mu X.\varphi \wedge \Box X$. It is easy to see that $\forall\varphi$ is satisfied at some state of a connected Kripke structure $\mathfrak{K}$ if and only if φ is satisfied at *all* states of $\mathfrak{K}$.

Let $\mathfrak{D}$ be a τ structure of bounded tree width, and let $Tt(\mathfrak{D})$ its tree representation as described in Sect. 4.1. We view $\mathcal{T}(\mathfrak{D})$ as a Kripke structure, with atomic propositions $\mathcal{O}_a$ and $R_{\boldsymbol{a}}$. Having available an universal modality, the axioms for tree representations $\mathcal{T}(\mathfrak{D})$, given in the previos subsection, can easily be expressed by modal formulae. For instance the consistency axioms can be written

$$\mathit{consistent}(\overline{R}) := \forall \bigwedge_{\boldsymbol{a}\in K^m} \Big(\bigwedge_{a\in\boldsymbol{a}} \mathcal{O}_a \wedge R_{\boldsymbol{a}} \rightarrow \Box\Big(\bigvee_{a\in\boldsymbol{a}} \neg\mathcal{O}_a \vee R_{\boldsymbol{a}}\Big)\Big).$$

Theorem 8. *Let $\mathfrak{D}$ be a structure with tree decomposition $\langle T, (\mathfrak{F}(v))_{v\in T}\rangle$. For an appropriate set of constants K and a function $f : D \rightarrow K$, let $\mathcal{T}(\mathfrak{D})$ be the associated tree structure. For every formula $\varphi(x_1, \dots, x_m)$ in μCGF and every tuple $\boldsymbol{a} \in K^m$ we can construct a formula $\varphi_{\boldsymbol{a}} \in L_\mu^{\leftarrow}$ such that, for every node v of T and every clique-guarded tuple $\boldsymbol{d} \subseteq \mathfrak{F}(v)$ with $f(\boldsymbol{d}) = \boldsymbol{a}$,*

$$\mathfrak{D} \models \varphi(\boldsymbol{d}) \iff \mathcal{T}(\mathfrak{D}), v \models \varphi_{\boldsymbol{a}}.$$

Proof. The translation is very similar to the translation into monadic second-order logic that was given in the previous section.

(1) If $\varphi(\boldsymbol{x})$ is an atom $Sx_{i_1}\cdots x_{i_m}$ then $\varphi_{\boldsymbol{a}} := S_{\boldsymbol{b}}$ where $\boldsymbol{b} = (a_{i_1}, \dots, a_{i_m})$.
(2) If $\varphi = (x_i = x_j)$, then $\varphi_{\boldsymbol{a}} := \mathit{true}$ if $a_i = a_j$ and $\varphi_{\boldsymbol{a}} := \mathit{false}$ otherwise.
(3) For the guard formulae $\mathit{clique}(\boldsymbol{x})$, let

$$\mathit{clique}_{\boldsymbol{a}} := \bigwedge_{a,a'\in\boldsymbol{a}} \mu X \,.\, \Big(\bigvee_{\substack{R\in\tau\\ \boldsymbol{b}:\, a,a'\in\boldsymbol{b}}} R_{\boldsymbol{b}} \vee \Diamond(\mathcal{O}_a \wedge \mathcal{O}_{a'} \wedge X)\Big).$$

(4) If $\varphi = \eta \wedge \vartheta$, then $\varphi_{\boldsymbol{a}} := \eta_{\boldsymbol{a}} \wedge \vartheta_{\boldsymbol{a}}$.
(5) If $\varphi = \neg\vartheta$, then $\varphi_{\boldsymbol{a}} := \neg\vartheta_{\boldsymbol{a}}$.
(6) If $\varphi = (\exists \boldsymbol{y} \,.\, \mathit{clique}(\boldsymbol{x}, \boldsymbol{y}))\eta(\boldsymbol{x}, \boldsymbol{y})$, then

$$\varphi_{\boldsymbol{a}} := \bigvee_{\boldsymbol{b}} \mu X \,.\, \Big(\Big(\bigwedge_{b\in\boldsymbol{b}} \mathcal{O}_b \wedge \mathit{clique}_{\boldsymbol{ab}} \wedge \eta_{\boldsymbol{ab}}\Big) \vee \Diamond\Big(\bigwedge_{a\in\boldsymbol{a}} \mathcal{O}_a \wedge X\Big)\Big).$$

(7) If $\varphi := [\mathrm{LFP}\ \ S\boldsymbol{x} \,.\, \eta(S, \boldsymbol{x})](\boldsymbol{x})$, let

$$\varphi_{\boldsymbol{a}} := [\mu\overline{S} \,.\, \overline{\eta}(\overline{S}))]_{\boldsymbol{a}}.$$

Here $\overline{S}$ is a tuple of fixed point variables $S_{\boldsymbol{b}}$ and $\overline{\eta}(\overline{S})$ is the tuple of the formulae $\eta_{\boldsymbol{b}}(\overline{S})$ for all $\boldsymbol{b} \in K^m$ (where m is the arity of S).

The proof that the translation is correct is analogous to the proof of Theorem 5. □

We now get another proof for the decidability of guarded fixed point logic. Given a sentence $\psi \in \mu$CGF, we translate it into the $L_\mu^{\leftarrow}$-sentence $\psi_\varnothing$ according to Theorem 8 and take the conjunction with the consistency axioms in $L_\mu^{\leftarrow}$ for tree representations $\mathcal{T}(\mathfrak{D})$. Then use Vardi's decidability result for $L_\mu^{\leftarrow}$. By the tree model property of $L_\mu^{\leftarrow}$, the tree model property of μCGF and Theorem 8 this gives a decision procedure for μCGF.

However, it is not clear whether this argument can modified to provide the optimal complexity bounds for guarded fixed point logic.

4.4 Alternating Tree Automata

Finally we sketch the automata theoretic method used in [6] for solving the satisfiability problem for guarded fixed point logic. The general idea is to associate with every sentence ψ an alternating automaton $\mathcal{B}_\psi$ that accepts precisely the (descriptions of) tree models of ψ. This reduces the satisfiability problem for ψ to the emptiness problem for $\mathcal{B}_\psi$, a problem that is solvable in exponential time with respect to the number of states of the automaton.

Automata based procedures are very important for the satisfiability testing and model checking of modal logics. For the μ-calculus and its relatives, *alternating tree automata* seem to be the right model to get optimal complexity results. In [12] Vardi used alternating *two-way* tree automata (A2A) to establish decidability and EXPTIME-completeness of the μ-calculus with backward modalities. His model of A2A works on trees of bounded branching where nodes have outgoing arcs labelled by indices $1, 2, \dots, k$. In our decidability proof for guarded fixed point logic in [6] we use a different variant of alternating automata which work on trees of arbitrary, finite or infinite, degree. The automata do not make use of the orientation of the edges in the tree; they may proceed to any neighbour of the current node, i.e. to the parent or to any child. A general forgetful determinacy theorem for parity games can be used to reduce the emptiness problem for our automata to the emptiness problem for Vardi's automata.

Definition 11. An alternating two-way automaton on trees is a tuple $\mathcal{A} = \langle Q_\exists, Q_\forall, \Sigma, q_0, \delta, \Omega \rangle$ where $Q = Q_\exists \cup Q_\forall$ is a finite set of states, partitioned into existential and universal states, Σ is an input alphabet and $q_0 \in Q$ is an initial state. The transition function has the form

$$\delta : Q \times \Sigma \to \mathcal{P}(Q \cup \{\circ q : q \in Q\})$$

Intuitively $q' \in \delta(q, \sigma)$ means that a copy of the automaton should stay at the current node and assume a new state q'. If $\circ q' \in \delta(q, \sigma)$ then a copy of the automaton should proceed to some neighbour of the current node and assume state q'. The function $\Omega : Q \to \mathbb{N}$ specifies the acceptance condition of the automaton.

Let $\mathcal{T} = \langle T, L : T \to \Sigma \rangle$ be a tree whose nodes are labelled by symbols from Σ. The notion of a run of $\mathcal{A}$ on $\mathcal{T}$ and the acceptance condition can be defined via a game $\mathcal{G}(\mathcal{A}, \mathcal{T})$, played by two players, player 0 and player 1, who move a token along the edges of a graph.

Let $\mathcal{G}(\mathcal{A}, \mathcal{T}) = \langle V_0, V_1, E, \Omega \rangle$ where

- $V_0 = T \times Q_\exists$ and $V_1 = T \times Q_\forall$;
- $(v, q)E(v', q')$ iff either
  - *(i)* $q' \in \delta(q, L(v))$ and $v' = v$, or
  - *(ii)* $\circ q' \in \delta(q, L(v))$ and v' is a neighbour of v.
- $\Omega(v, q) = \Omega(q)$.

Initially, the token is at the vertex (λ, q_0) where λ is the root of T and q_0 is the initial state of the automaton. When the token is at a vertex $(v,q) \in V_j$, then player j moves it along some edge $(v,q)E(v',q')$ to a new node (v',q'). If one of the players cannot make a move she looses. Otherwise the result of the game is an infinite path $(\lambda, q_0), (v_1, q_1), (v_2, q_2), \ldots$ through the graph. In that case player 0 wins iff the sequence $\Omega(q_0), \Omega(q_1), \Omega(q_2), \ldots$ satisfies the parity condition: the minimal number $i \in \mathbb{N}$ appearing infinitely often in the sequence is even. (Therefore such games are called parity games.) By definition the automaton $\mathcal{A}$ accepts $\mathcal{T}$ if and only if player 0 has a winning strategy from position (λ, q_0) in $\mathcal{G}(\mathcal{A}, \mathcal{T})$.

Definition 12. A *memoryless strategy* for player 0 is a partial function $f : (T \times Q_\exists) \to T \times Q$ such that $(v,q)E(v',q')$ whenever $f(v,q) = (v',q')$. A finite or infinite play $(w_1,q_1),(w_2,q_2),\ldots$ of $\mathcal{G}(\mathcal{A},\mathcal{T})$ is *consistent with* f if for every pair (w_i,q_i) such that $q_i \in Q_\exists$ the value of $f(w_i,q_i)$ is defined and $(w_{i+1},q_{i+1}) = f(w_i,q_i)$. A memoryless strategy is *winning* from position (v,q) if player 0 wins every play starting at (v,q) that is consistent with f.

For a large class of games, there are *forgetful determinacy theorems* which say that from each position, one of the two players has a memoryless winning strategy. In particular this is the case for the graph games with parity winning conditions that we use here (see e.g. [13, Theorem 1]).

Theorem 9. *The automaton $\mathcal{A}$ accepts $\mathcal{T}$ if and only there exists a memoryless winning strategy for player 0 on $\mathcal{G}(\mathfrak{A}, \mathcal{T})$ from the initial position (λ, q_0).*

As a consequence, one can derive the following results (see [6] for further details).

Theorem 10. *If an alternating two-way automaton accepts some input tree, then it also accepts a tree whose branching is bounded by the number of states of the automaton. The emptiness problem for alternating two-way automata (on trees with arbitrary branching) can be decided in* Exptime.

Accepting Tree Models by Automata. In [6] we construct for every guarded fixed point sentence ψ an automaton $\mathcal{A}_\psi$ accepting exactly those tableaux that represent a model for ψ. We do not explain the notion of a tableau here. Instead we present an almost identical construction of an automaton $\mathcal{B}_\psi$ that accepts a tree $\mathcal{T}$ if and only if $\mathcal{T}$ is a tree representation (more or less as in Sect 4.1) of a model $\mathfrak{D} \models \psi$.

Let $\langle T, (\mathfrak{F}(v))_{v\in T}\rangle$ be a tree decomposition of width $k-1$ of a τ-structure $\mathfrak{D}$ with universe D, let K be a set of $2k$ constants and let $f : D \to K$ be an appropriate function naming each element d of $\mathfrak{D}$ by a constant $a_d \in K$ as described in Sect. 4.1. Our automaton works on the description of $\langle T, (\mathfrak{F}(v))_{v\in T}\rangle$ by a labelled tree $\langle T, L\rangle$, where the label of $L(v)$ of a node v is the pair (C, Γ) such that $C \subseteq K$ is the set of constants that occur at the node v, and Γ contains all atomic and negated atomic formulae $\beta(\boldsymbol{a})$ and all guard formulae *clique*$(\boldsymbol{a})$

that are true at node v. It is not difficult to design an alternating automaton that checks whether a given tree is consistently labelled in this sense. One could also integrate the checking of these conditions directly into the automaton $\mathcal{B}_\psi$ constructed below but this would make the construction less transparent.

Let ψ be a sentence in μCGF of vocabulary τ which is in positive normal form (i.e. with negations only in front of atomic formulae). We assume that the fixed point variables in ψ are $T_1, \dots, T_n$, and that for each fixed point variable T_i there is only one formula of the form $[\text{FP } T_i\boldsymbol{x} \,.\, \eta(T_i, \boldsymbol{x})](\boldsymbol{x})$ in ψ (where FP is either LFP or GFP; accordingly T_i is either an LFP-variable or a GFP-variable). Further we assume that for $j > i$, T_j does not preceed T_i in the dependency order of the fixed point variables in ψ, i.e. T_j does not occur free in $[\text{FP } T_i\boldsymbol{x} \,.\, \eta(T_i, \boldsymbol{x})](\boldsymbol{x})$.

Our goal is to describe an automaton $\mathcal{B}_\psi$ that, given a consistently labelled tree $\langle T, L\rangle$ describing a structure $\mathfrak{D}$, checks whether $\mathfrak{D} \models \psi$.

Definition 13. We denote by $cl(\psi)$ the smallest set of formulae such that all subformulae of ψ belong to $cl(\psi)$, and for every relation symbol R occurring in ψ (including equality), $cl(\psi)$ contains formulae $Rx_1 \cdots x_k$ and $\neg Rx_1 \cdots x_k$ with distinct variable symbols $x_1, \dots, x_k$. Further, we put

$$cl(\psi, K) := \{\varphi(\boldsymbol{a}) : \boldsymbol{a} \subseteq K, \varphi(\boldsymbol{x}) \in cl(\psi)\} \cup \{\mathit{true}, \mathit{false}\}.$$

The states of the automaton $\mathcal{B}_\psi$ will be the formulae in $cl(\psi, K)$, with initial state ψ, and the following partition into existential and universal states:

$$\begin{aligned} Q_\exists &= \{\varphi \in cl(\psi, K) : \varphi = \eta \vee \vartheta \text{ or } \varphi = \exists \boldsymbol{x}.\eta\} \cup \{\mathit{false}\} \\ Q_\forall &= cl(\psi, K) - Q_\exists. \end{aligned}$$

The transition function δ is defined as follows.

(1) $\delta(\mathit{true}, (C, \Gamma)) = \delta(\mathit{false}, (C, \Gamma)) = \varnothing$.

(2) If α is a τ-atom or a negated τ-atom then

$$\delta(\alpha, (C, \Gamma)) = \begin{cases} \mathit{true} & \text{if } \alpha \in \Gamma \\ \mathit{false} & \text{if } \alpha \notin \Gamma \end{cases}$$

(3) $\delta(\varphi \vee \vartheta, (C, \Gamma)) = \delta(\varphi \wedge \vartheta, (C, \Gamma)) = \{\varphi, \vartheta\}$.

(4) $\delta([\text{FP } T\boldsymbol{x} \,.\eta(T, \boldsymbol{x})](\boldsymbol{a}), (C, \Gamma)) = \{\eta(T, \boldsymbol{a})\}$.

(5) $\delta(T\boldsymbol{a}, (C, \Gamma)) = \{\eta(T, \boldsymbol{a})\}$ where $[\text{FP } T\boldsymbol{x}.\eta(T, \boldsymbol{x})]$ is the unique fixed point formula in ψ that binds T.

(6) Let φ be of the form $(\exists \boldsymbol{x} \,.\, \mathit{clique}(\boldsymbol{a}, \boldsymbol{x}))\eta(\boldsymbol{a}, \boldsymbol{x})$ or $(\forall \boldsymbol{x} \,.\, \mathit{clique}(\boldsymbol{a}, \boldsymbol{x}))\eta(\boldsymbol{a}, \boldsymbol{x})$. Then

$$\delta(\varphi, (C, \Gamma)) = \begin{cases} \{\eta(\boldsymbol{a}, \boldsymbol{b}) : \boldsymbol{b} \subseteq C, \mathit{clique}(\boldsymbol{a}, \boldsymbol{b}) \in \Gamma\} \cup \{\circ\varphi\} & \text{if } \boldsymbol{a} \subseteq C \\ \varnothing & \text{if } \boldsymbol{a} \not\subseteq C. \end{cases}$$

The acceptance condition Ω is defined by

$$\Omega(\varphi) = \begin{cases} 2i-1 & \text{for } \varphi = T_i\boldsymbol{a} \text{ and } T_i \text{ an LFP-variable} \\ 2i & \text{for } \varphi = T_i\boldsymbol{a} \text{ and } T_i \text{ a GFP-variable} \\ 2n+1 & \text{for } \varphi = \exists \boldsymbol{x} \,.\, \eta \\ 2n+2 & \text{otherwise.} \end{cases}$$

Theorem 11. *Suppose that $\langle T, L\rangle$ represents a tree-decomposition of width $k-1$ of a structure $\mathfrak{D}$. Then $\mathfrak{B}_\psi$ accepts $\langle T, L\rangle$ if and only if $\mathfrak{D} \models \psi$.*

We just try to convey some intuition why this is true. For a detailed proof, we refer to [6].

The game $\mathcal{G}(\mathcal{B}_\psi, \mathcal{T})$ associated with the automaton $\mathfrak{B}_\psi$ on input $\mathcal{T}$ is basically just the *model checking game* for ψ on the structure $\mathfrak{D}$ that is represented by $\mathcal{T}$. (This game generalizes the model checking game for the μ-calculus described for instance in [10]). It is not difficult to see that if player 0 has a strategy to win $\mathcal{G}(\mathcal{B}_\psi, \mathcal{T})$ in a *finite* number of moves, then $\mathfrak{D} \models \psi$, and if player 1 has such a strategy, then $\mathfrak{D} \models \neg\psi$. However, the interesting plays are the infinite ones. We have to distinguish different cases:

(1) Suppose that there are infinitely many occurrences of fixed point formulae $T\boldsymbol{a}$ in the play. Then the 'outermost' fixed point variable T_i (the least with respect to the dependency order) that appears infinitely often determines which player has won. If it is a LFP-variable then the minimal number $\Omega(q)$ appearing infinitely often on the play is odd and therefore the play is winning for player 1. One can show that if player 1 has a strategy to enforce this, then $\mathfrak{D} \models \neg\psi$ since the regeneration sequence of some LFP-formula in ψ is not well-founded. If T_i is a GFP-variable, then player 0 wins and $\mathfrak{D} \models \psi$.
(2) If no fixed point variable appears infinitely often on a infinite play, then there is an existential or universal formula that is never reduced but always 'passed over' to a neighbouring node (see clause (6) of the definition of the automaton). If this formula is existential, say $\exists \boldsymbol{x}.\eta(\boldsymbol{x})$, then by the definition of Ω, player 0 looses. Indeed, if she gets trapped in such a situation then this means that she cannot find a witness $\boldsymbol{a}$ and then win the remaining game from $\eta(\boldsymbol{a})$. Hence $\mathfrak{D} \models \neg\psi$ (recall that ψ is in positive normal form). If the formula is of the form $\forall x.\eta(\boldsymbol{x})$, then player 1 wins. Indeed, such an outcome just means that player 1 does not find a 'bad' $\boldsymbol{a}$ such that he hopes to win on $\eta(\boldsymbol{a})$. In this case $\mathfrak{D} \models \psi$.

Note that the number of states of the automaton B_ψ is bounded by $|\psi|^{2k\log k}$ where k is the width of ψ. Given that the emptiness problem for our automata can be solved in exponential time with respect to the number of states, it follows that the satisfiability problem for guarded fixed point logic is solvable in 2EXPTIME for sentences of unbounded width an in EXPTIME for sentences of bounded width.

References

1. H. Andréka, J. van Benthem, and I. Németi, *Modal languages and bounded fragments of predicate logic*, Journal of Philosophical Logic, 27 (1998), pp. 217–274.
2. J. van Benthem, *Dynamic bits and pieces*, ILLC research report, University of Amsterdam, 1997.
3. A. Emerson and C. Jutla, *The complexity of tree automata and logics of programs*, in Proc. 29th IEEE Symp. on Foundations of Computer Science, 1988, pp. 328–337.
4. J. Flum, *On the (infinite) model theory of fixed-point logics*, in Models, algebras, and proofs, X. Caicedo and C. Montenegro, eds., no. 2003 in Lecture Notes in Pure and Applied Mathematics Series, Marcel Dekker, 1999, pp. 67–75.
5. E. Grädel, *On the restraining power of guards*, Journal of Symbolic Logic. To appear.
6. E. Grädel and I. Walukiewicz, *Guarded fixed point logic*, in Proc. 14th IEEE Symp. on Logic in Computer Science, 1999.
7. D. Kozen, *Results on the propositional μ-calculus*, Theoretical Computer Science, 27 (1983), pp. 333–354.
8. M. Rabin, *Decidability of second-order theories and automata on infinite trees*, Transactions of the AMS, 141 (1969), pp. 1–35.
9. B. Reed, *Tree width and tangles: A new connectivity measure and some applications*, in Surveys in Combinatorics, R. Bailey, ed., Cambridge University Press, 1997, pp. 87–162.
10. C. Stirling, *Bisimulation, model checking and other games.* Notes for the Mathfit instructional meeting on games and computation. Edinburgh, 1997.
11. M. Vardi, *Why is modal logic so robustly decidable?*, in Descriptive Complexity and Finite Models, N. Immerman and P. Kolaitis, eds., vol. 31 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS, 1997, pp. 149–184.
12. ———, *Reasoning about the past with two-way automata*, in Automata, Languages and Programming ICALP 98, vol. 1443 of Springer Lecture Notes in Computer Science, 1998, pp. 628–641.
13. W. Zielonka, *Infinite games on finitely coloured graphs with applications to automata on infinite trees*, Theoretical Computer Science, 200 (1998), pp. 135–183.

A PSpace Algorithm for Graded Modal Logic

Stephan Tobies[*]

LuFg Theoretical Computer Science, RWTH Aachen
tobies@informatik.rwth-aachen.de

Abstract. We present a PSPACE algorithm that decides satisfiability of the graded modal logic $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$—a natural extension of propositional modal logic $\mathbf{K}_{\mathcal{R}}$ by counting expressions—which plays an important role in the area of knowledge representation. The algorithm employs a tableaux approach and is the first known algorithm which meets the lower bound for the complexity of the problem. Thus, we exactly fix the complexity of the problem and refute a EXPTIME-hardness conjecture. This establishes a kind of "theoretical benchmark" that all algorithmic approaches can be measured with.

1 Introduction

Propositional modal logics have found applications in many areas of computer science. Especially in the area of knowledge representation, the description logic (DL) $\mathcal{ALC}$, which is a syntactical variant of the propositional (multi-)modal logic $\mathbf{K}_{\mathcal{R}}$ [Sch91], forms the basis of a large number of formalisms used to represent and reason about conceptual and taxonomical knowledge of the application domain. The graded modal logic $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ extends $\mathbf{K}_{\mathcal{R}}$ by *graded modalities* [Fin72], i.e., counting expressions which allow to express statements of the form "there are at least (at most) n accessible worlds that satisfy ... ". This is especially useful in knowledge representation because (a) humans tend to describe objects by the number of other objects they are related to (a stressed person is a person given at least three assignments that are urgent), and (b) qualifying number restrictions (the DL's analogue for graded modalities [HB91]) are necessary for modeling semantic data models [CLN94].

$\mathbf{K}_{\mathcal{R}}$ is decidable in PSPACE and can be embedded into a decidable fragment of predicate logic [AvBN98]. Hence, there are two general approaches for reasoning with $\mathbf{K}_{\mathcal{R}}$: dedicated decision procedures [Lad77,SSS91,GS96], and the translation into first order logic followed by the application of an existing first order theorem prover [OS97,Sch97]. To compete with the dedicated algorithms, the second approach has to yield a decision procedure and it has to be efficient, because the dedicated algorithms usually have optimal worst-case complexity. For $\mathbf{K}_{\mathcal{R}}$, the first issue is solved and, regarding the complexity, experimental results show that the algorithm competes well with dedicated algorithms [HS97]. Since experimental result can only be partially satisfactory, a theoretical complexity

[*] This work was supported by the DFG, Project No. GR 1324/3–1

result would be desirable, but there are no exact results on the complexity of the theorem prover approach.

The situation for $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ is more complicated: $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ is known to be decidable, but this result is rather recent [HB91], and the known PSpace upper complexity bound for $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ is only valid if we assume unary coding of numbers in the input, which is an unnatural restriction. For binary coding no upper bound is known and the problem has been conjectured to be ExpTime-hard [dHR95]. This coincides with the observation that a straightforward adaption of the translation technique leads to an exponential blow-up in the size of the first order formula. This is because it is possible to store the number n in $\log_k n$-bits if numbers are represented in k-ary coding. In [OSH96] a translation technique that overcomes this problem is proposed, but a decision procedure for the target fragment of first order logic yet has to be developed.

In this work we show that reasoning for $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ is not harder than reasoning for $\mathbf{K}_{\mathcal{R}}$ by presenting an algorithm that decides satisfiability in PSpace, even if the numbers in the input are binary coded. It is based on the tableaux algorithms for $\mathbf{K}_{\mathcal{R}}$ and tries to prove the satisfiability of a given formula by explicitly constructing a model for it. When trying to generalise the tableaux algorithms for $\mathbf{K}_{\mathcal{R}}$ to deal with $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$, there are some difficulties: (1) the straightforward approach leads to an incorrect algorithm; (2) even if this pitfall is avoided, special care has to be taken in order to obtain a space-efficient solution. As an example for (1), we will show that the algorithm presented in [dHR95] to decide satisfiability of $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ is incorrect. Nevertheless, this algorithm will be the basis of our further considerations. Problem (2) is due to the fact that tableaux algorithms try to prove the satisfiability of a formula by explicitly building a model for it. If the tested formula requires the existence of n accessible worlds, a tableaux algorithm will include them in the model it constructs, which leads to exponential space consumption, at least if the numbers in the input are not unarily coded or memory is not re-used. An example for a correct algorithm which suffers from this problem can be found in [HB91] and is briefly presented in this paper. Our algorithm overcomes this problem by organising the search for a model in a way that allows for the re-use of space *for each successor*, thus being capable of deciding satisfiability of $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ in PSpace.

2 Preliminaries

In this section we introduce the graded modal logic $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$, the extension of the multi-modal logic $\mathbf{K}_{\mathcal{R}}$ with graded modalities, first introduced in [Fin72].

Definition 1 (Syntax and Semantics of $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$). *Let $\mathcal{P} = \{p_0, p_1, \dots\}$ be a set of propositional atoms and $\mathcal{R}$ a set of* relation names. *The set of* $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$-formulae *is built according to the following rules:*

1. *every propositional atom is a $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$-formula, and*
2. *if ϕ, ψ_1, ψ_2 are $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$-formulae, $n \in \mathbb{N}$, and R is a relation name, then $\neg\phi$, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\langle R \rangle_n \phi$, and $[R]_n \phi$ are formulae.*

The semantics of $\mathbf{Gr}(\mathbf{K}_\mathcal{R})$*-formulae is based on* Kripke structures

$$\mathfrak{M} = (W^\mathfrak{M}, \{R^\mathfrak{M} \mid R \in \mathcal{R}\}, V^\mathfrak{M}),$$

where $W^\mathfrak{M}$ *is a non-empty set of worlds, each* $R^\mathfrak{M} \subseteq W^\mathfrak{M} \times W^\mathfrak{M}$ *is an* accessibility relation *on worlds (for* $R \in \mathcal{R}$*), and* $V^\mathfrak{M}$ *is a* valuation *assigning subsets of* $W^\mathfrak{M}$ *to the propositional atoms in* $\mathcal{P}$*. For a Kripke structure* $\mathfrak{M}$*, an element* $x \in W^\mathfrak{M}$*, and a* $\mathbf{Gr}(\mathbf{K}_\mathcal{R})$*-formula, the model relation* $\models$ *is defined inductively on the structure of formulae:*

$$\begin{aligned}
&\mathfrak{M}, x \models p \textit{ iff } x \in V^\mathfrak{M}(p) \textit{ for } p \in \mathcal{P}\\
&\mathfrak{M}, x \models \neg\phi \textit{ iff } \mathfrak{M}, x \not\models \phi\\
&\mathfrak{M}, x \models \psi_1 \wedge \psi_2 \textit{ iff } \mathfrak{M}, x \models \psi_1 \textit{ and } \mathfrak{M}, x \models \psi_2\\
&\mathfrak{M}, x \models \psi_1 \vee \psi_2 \textit{ iff } \mathfrak{M}, x \models \psi_1 \textit{ or } \mathfrak{M}, x \models \psi_2\\
&\mathfrak{M}, x \models \langle R\rangle_n\phi \textit{ iff } \sharp R^\mathfrak{M}(x,\phi) > n\\
&\mathfrak{M}, x \models [R]_n\phi \textit{ iff } \sharp R^\mathfrak{M}(x,\neg\phi) \leq n
\end{aligned}$$

where $\sharp R^\mathfrak{M}(x,\phi) := |\{y \in W^\mathfrak{M} \mid (x,y) \in R^\mathfrak{M} \textit{ and } \mathfrak{M}, y \models \phi\}|$

The propositional modal logic $\mathbf{K}_\mathcal{R}$ *is defined as the fragment of* $\mathbf{Gr}(\mathbf{K}_\mathcal{R})$ *in which for all modalities* $n = 0$ *holds.*

A formula is called satisfiable *iff there exists a structure* $\mathfrak{M}$ *and a world* $x \in W^\mathfrak{M}$ *such that* $\mathfrak{M}, x \models \phi$*.*

By $\mathrm{SAT}(\mathbf{Gr}(\mathbf{K}_\mathcal{R}))$*and* $\mathrm{SAT}(\mathbf{K}_\mathcal{R})$*we denote the sets of satisfiable formulae of* $\mathbf{Gr}(\mathbf{K}_\mathcal{R})$ *and* $\mathbf{K}_\mathcal{R}$*, respectively.*

As usual, the modalities $\langle R\rangle_n\phi$ and $[R]_n\phi$ are dual: $\sharp R^\mathfrak{M}(x,\phi) > n$ means that in $\mathfrak{M}$ more than n R-successors of x satisfy ϕ; $\sharp R^\mathfrak{M}(x,\neg\phi) \leq n$ means that in $\mathfrak{M}$ all but at most n R-successors satisfy ϕ.

In the following we will only consider formulae in *negation normal form* (NNF), a form in which negations have been pushed inwards and occur in front of propositional atoms only. We will denote the NNF of $\neg\phi$ by $\sim\phi$. The NNF can always be generated in linear time and space by successively applying the following equivalences from left to right:

$$\begin{aligned}
\neg(\psi_1 \wedge \psi_2) &\equiv \neg\psi_1 \vee \neg\psi_2 \qquad & \neg\langle R\rangle_n\psi &\equiv [R]_n\neg\psi\\
\neg(\psi_1 \vee \psi_2) &\equiv \neg\psi_1 \wedge \neg\psi_2 \qquad & \neg[R]_n\psi &\equiv \langle R\rangle_n\neg\psi
\end{aligned}$$

3 Reasoning for $\mathbf{Gr}(\mathbf{K}_\mathcal{R})$

Before we present our algorithm for deciding satisfiability of $\mathbf{Gr}(\mathbf{K}_\mathcal{R})$, for historic and didactic reasons, we present two other solutions: an incorrect one [dHR95], and a solution that is less efficient [HB91].

From the fact that $\mathrm{SAT}(\mathbf{K}_\mathcal{R})$is PSPACE-complete [Lad77,HM92], it immediately follows, that $\mathrm{SAT}(\mathbf{Gr}(\mathbf{K}_\mathcal{R}))$is PSPACE-hard. The algorithms we will consider decide the satisfiability of a given formula ϕ by trying to construct a model for ϕ.

3.1 An Incorrect Algorithm

In [dHR95], an algorithm for deciding SAT($\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$)is given, which, unfortunately, is incorrect. Nevertheless, it will be the basis for our further considerations and thus it is presented here. It will be referred to as the *incorrect* algorithm. It is based on an algorithm given in [DLNN97] to decide the satisfiability of the DL $\mathcal{ALCNR}$, which basically is the restriction of $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$, where, in formulae of the form $\langle R\rangle_n\phi$ or $[R]_n\phi$ with $n > 0$, necessarily $\phi = p \vee \neg p$ holds.

The algorithm for $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ tries to build a model for a formula ϕ by manipulating sets of constraints with the help of so-called *completion rules.* This is a well-known technique to check the satisfiability of modal formulae, which has already been used to prove decidability and complexity results for other DLs (e. g., [SSS91,HB91,BBH96]). These algorithms can be understood as variants of tableaux algorithms which are used, for example, to decide satisfiability of the modal logics $\mathbf{K}_{\mathcal{R}}$, $\mathbf{T}_{\mathcal{R}}$, or $\mathbf{S4}_{\mathcal{R}}$ in [HM92].

Definition 2. *Let $\mathcal{V}$ be a set of variables. A* constraint system *(c.s.) S is a finite set of expressions of the form '$x \models \phi$' and 'Rxy', where ϕ is a formula, $R \in \mathcal{R}$, and $x, y \in \mathcal{V}$.*

For a c.s. S, let $\sharp R^S(x,\phi)$ be the number of variables y for which $\{Rxy, y \models \phi\} \subseteq S$. The c.s. $[z/y]S$ is obtained from S by replacing every occurrence of y by z; this replacement is said to be safe *iff, for every variable x, formula ϕ, and relation symbol R with $\{x \models \langle R\rangle_n\phi, Rxy, Rxz\} \subseteq S$ we have $\sharp R^{[z/y]S}(x,\phi) > n$.*

A c.s. S is said to contain a clash *iff for a propositional atom p, a formula ϕ, and $m \leq n$:*

$$\{x \models p, x \models \neg p\} \subseteq S \text{ or } \{x \models \langle R\rangle_m\phi, x \models [R]_n{\sim}\phi\} \subseteq S.$$

Otherwise it is called clash-free. *A c.s. S is called* complete *iff none of the rules given in Fig. 1 are applicable to S.*

To test the satisfiability of a formula ϕ, the incorrect algorithm works as follows: It starts with the c.s. $\{x \models \phi\}$ and successively and applies the rules given in Fig. 1, stopping if a clash is occurs. Both the selection of the rule to apply and the selection of the formula to add (in the the $\rightarrow_\vee$-rule) or the variables to identify (in the $\rightarrow_\leq$-rule) are selected non-deterministically. The algorithm answers "ϕ is satisfiable" iff the rules can be applied in a way that yields a complete and clash-free c.s. The notion of *safe* replacement of variables is needed to ensure the termination of the rule application [HB91].

Since we are interested in PSpace algorithms, non-determinism imposes no problem due to Savitch's Theorem, which states that deterministic and non-deterministic polynomial space coincide [Sav70].

To prove the correctness of a non-deterministic completion algorithm, it is sufficient to prove three properties of the model generation process:

1. Termination: Any sequence of rule applications is finite.
2. Soundness: If the algorithm terminates with a complete and clash-free c.s. S, then the tested formula is satisfiable.

$\rightarrow_\wedge$-rule: if 1. $x \models \psi_1 \wedge \psi_2 \in S$ and
2. $\{x \models \psi_1, x \models \psi_2\} \not\subseteq S$
then $S \rightarrow_\wedge S \cup \{x \models \psi_1, x \models \psi_2\}$

$\rightarrow_\vee$-rule: if 1. $(x \models \psi_1 \vee \psi_2) \in S$ and
2. $\{x \models \psi_1, x \models \psi_2\} \cap S = \emptyset$
then $S \rightarrow_\vee S \cup \{x \models \chi\}$ where $\chi \in \{\psi_1, \psi_2\}$

$\rightarrow_>$-rule: if 1. $x \models \langle R \rangle_n \phi \in S$ and
2. $\sharp R^S(x, \phi) \leq n$
then $S \rightarrow_> S \cup \{Rxy, y \models \phi\}$ where y is a fresh variable.

$\rightarrow_{\leq 0}$-rule: if 1. $x \models [R]_0 \phi, Rxy \in S$ and
2. $y \models \phi \notin S$
then $S \rightarrow_{\leq 0} S \cup \{y \models \phi\}$

$\rightarrow_\leq$-rule: if 1. $x \models [R]_n \phi, \sharp R^S(x, \phi) > n > 0$ and
2. $Rxy, Rxz \in S$ and
3. replacing y by z is safe in S
then $S \rightarrow_\leq [z/y]S$

Fig. 1. The incorrect completion rules for $\mathbf{Gr}(\mathbf{K}_\mathcal{R})$.

3. Completeness: If the formula is satisfiable, then there is a sequence of rule applications that yields a complete and clash-free c.s.

The error of the incorrect algorithm is, that is does not satisfy Property 2, even though the converse is claimed:

CLAIM([dHR95]): Let ϕ be a $\mathbf{Gr}(\mathbf{K}_\mathcal{R})$-formula in NNF. ϕ is satisfiable iff $\{x_0 \models \phi\}$ can be transformed into a clash-free complete c.s. using the rules from Figure 1.

Unfortunately, the *if*-direction of this claim is not true, which we will prove by a simple counterexample. Consider the formula

$$\phi = \langle R \rangle_2 p_1 \wedge [R]_1 p_2 \wedge [R]_1 \neg p_2.$$

On the one hand, ϕ is not satisfiable. Assume $\mathfrak{M}, x \models \langle R \rangle_2 p_1$. This implies the existence of at least three R-successors y_1, y_2, y_3 of x. For each of the y_i either $\mathfrak{M}, y_i \models p_2$ or $\mathfrak{M}, y_i \not\models p_2$ holds by the definition of $\models$. Without loss of generality, there are two worlds y_{i_1}, y_{i_2} such that $\mathfrak{M}, y_{i_j} \models p_2$, which implies $\mathfrak{M}, x \not\models [R]_1 \neg p_2$ and hence $\mathfrak{M}, x \not\models \phi$.

On the other hand, the c.s. $S = \{x \models \phi\}$ can be turned into a complete and clash-free c.s. using the rules from Fig. 1, as is shown in Fig. 2. Clearly this invalidates the proof of the claim.

3.2 An Alternative Syntax

At this stage the reader may have noticed the cumbersome semantics of the $[R]_n$ modality, which origins from the wish that the duality $\Box\phi \equiv \neg\Diamond\neg\phi$ of $\mathbf{K}$ carries

$$\{x \models \phi\} \rightarrow_{\wedge} \cdots \rightarrow_{\wedge} \underbrace{\{x \models \phi,\ x \models \langle R\rangle_2 p_1,\ x \models [R]_1 p_2,\ x \models [R]_1 \neg p_2\}}_{=S_1}$$
$$\rightarrow_{>} \cdots \rightarrow_{>} \underbrace{S_1 \cup \{Rxy_i,\ y_i \models p_1 \mid i = 1, 2, 3\}}_{=S_2}$$

S_2 is clash-free and complete, because $\sharp R^{S_2}(x, p_1) = 3$ and $\sharp R^{S_2}(x, p_2) = 0$.

Fig. 2. A run of the incorrect algorithm.

over to $[R]_n\phi \equiv \neg\langle R\rangle_n\neg\phi$ in $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$. This makes the semantics of $[R]_n$ and $\langle R\rangle_n$ un-intuitive. Not only does the n in a diamond modality mean "more than n" while it means "less *or equal* than n" for a box modality. The semantics also introduce a "hidden" negation.

To overcome these problems, we will replace these modalities by a syntax inspired by the counting quantifiers in predicate logic: the modalities $\langle R\rangle_{\leq n}$ and $\langle R\rangle_{\geq n}$ with semantics defined by :

$$\mathfrak{M}, x \models \langle R\rangle_{\leq n}\phi \text{ iff } \sharp R^{\mathfrak{M}}(x, \phi) \leq n,$$
$$\mathfrak{M}, x \models \langle R\rangle_{\geq n}\phi \text{ iff } \sharp R^{\mathfrak{M}}(x, \phi) \geq n.$$

This modification does not change the expressivity of the language, since $\mathfrak{M}, x \models \langle R\rangle_n\phi$ iff $\mathfrak{M}, x \models \langle R\rangle_{\geq n-1}\phi$ and $\mathfrak{M}, x \models [R]_n\phi$ iff $\mathfrak{M}, x \models \langle R\rangle_{\leq n}\neg\phi$.

3.3 A Correct but Inefficient Solution

To understand the mistake of the incorrect algorithm, it is useful to known how soundness is usually established for the kind of algorithms we consider. The underlying idea is that a complete and clash-free c.s. induces a model for the formula tested for satisfiability:

Definition 3 (Canonical Structure). *Let S be a c.s. The* canonical structure $\mathfrak{M}_S = (W^{\mathfrak{M}_S}, \{R^{\mathfrak{M}_S} \mid R \in \mathcal{R}\}, V^{\mathfrak{M}_S})$ induced by S *is defined as follows:*

$$W^{\mathfrak{M}_S} = \{x \in \mathcal{V} \mid x \textit{ occurs in } S\},$$
$$R^{\mathfrak{M}_S} = \{(x, y) \in \mathcal{V}^2 \mid Rxy \in S\},$$
$$V^{\mathfrak{M}_S}(p) = \{x \in \mathcal{V} \mid x \models p \in S\}.$$

Using this definition, it is then easy to prove that the canonical structure induced by a complete and clash-free c.s. is a model for the tested formula.

The mistake of the incorrect algorithm is due to the fact that it did not take into account that, in the canonical model induced by a complete and clash-free c.s., there are formulae satisfied by the worlds even though these formulae do not appear as constraints in the c.s. Already in [HB91], an algorithm very similar

$\rightarrow_\wedge$-, $\rightarrow_\vee$-rule: see Fig. 1

$\rightarrow_{\text{choose}}$-rule: if 1. $x \models \langle R \rangle_{\bowtie n} \phi, Rxy \in S$ and
2. $\{y \models \phi, y \models \sim\phi\} \cap S = \emptyset$
then $S \rightarrow_{\text{choose}} S \cup \{y \models \chi\}$ where $\chi \in \{\phi, \sim\phi\}$

$\rightarrow_{\geq}$-rule: if 1. $x \models \langle R \rangle_{\geq n} \phi \in S$ and
2. $\sharp R^S(x, \phi) < n$
then $S \rightarrow_{\geq} S \cup \{Rxy, y \models \phi\}$ where y is a new variable.

$\rightarrow_{\leq}$-rule: if 1. $x \models \langle R \rangle_{\leq n} \phi, \sharp R^S(x, \phi) > n$ and
2. $y \neq z, Rxy, Rxz, y \models \phi, z \models \phi \in S$ and
3. the replacement of y by z is safe in S
then $S \rightarrow_{\leq} [y/z]S$

Fig. 3. The standard completion rules

to the incorrect one is presented which decides the satisfiability of $\mathcal{ALCQ}$, a notational variant of $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$.

The algorithm essentially uses the same definitions and rules. The only differences are the introduction of the $\rightarrow_{\text{choose}}$-rule and an adaption of the $\rightarrow_{\geq}$-rule to the alternative syntax. The $\rightarrow_{\text{choose}}$-rule makes sure that all "relevant" formulae that are implicitly satisfied by a variable are made explicit in the c.s. Here, relevant formulae for a variable y are those occuring in modalities in constraints for variables x such that Rxy appears in the c.s. The complete rule set for the modified syntax of $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ is given in Fig. 3. The definition of *clash* has to be modified as well: A c.s. S contains a clash iff

- $\{x \models p, x \models \neg p\} \subseteq S$ for some variable x and a propositional atom p, or
- $x \models \langle R \rangle_{\leq n} \phi \in S$ and $\sharp R^S(x, \phi) > n$ for some variable x, relation R, formula ϕ, and $n \in \mathbb{N}$.

The algorithm, which works like the incorrect algorithm but uses the expansion rules from Fig. 3 and the definition of clash from above will be called the *standard algorithm*; it is a decision procedure for SAT($\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$):

Theorem 1 ([HB91]). *Let ϕ be a $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$-formula in NNF. ϕ is satisfiable iff $\{x_0 \models \phi\}$ can be transformed into a clash-free complete c.s. using the rules in Figure 3. Moreover, each sequence of these rule-applications is finite.*

While no complexity result is explicitly given in [HB91], it is easy to see that a PSpace result could be derived from the algorithm using the trace technique, employed in [SSS91] to show that satisfiability of $\mathcal{ALC}$, the notational variant for $\mathbf{K}_{\mathcal{R}}$, is decidable in PSpace.

Unfortunately this is only true if we assume the numbers in the input to be unary coded. The reason for this lies in the $\rightarrow_{\geq}$-rule, which generates n successors for a formula of the form $\langle R \rangle_{\geq n} \phi$. If n is unary coded, these successors consume at least polynomial space in the size of the input formula. If we assume binary (or k-ary with $k > 1$) encoding, the space consumption is exponential in the size of the input because a number n can be represented in $\log_k n$ bits in k-ary coding. This blow-up can not be avoided because the completeness of the standard algorithm relies on the generation *and identification* of these successors, which makes it necessary to keep them in memory *at one time*.

4 An Optimal Solution

In the following, we will present the algorithm which will be used to prove the following theorem; it contrasts the EXPTIME-hardness conjecture in [dHR95].

Theorem 2. *Satisfiability for* $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ *is* PSPACE-*complete if numbers in the input are represented using* ***binary*** *coding.*

When aiming for a PSPACE algorithm, it is impossible to generate all successors of a variable in a c.s. at a given stage because this may consume space that is exponential in the size of the input concept. We will give an optimised rule set for $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$-satisfiability that does not rely on the identification of successors. Instead we will make stronger use of non-determinism to guess the assignment of the relevant formulae to the successors by the time of their generation. This will make it possible to generate the c.s. in a depth first manner, which will facilitate the re-use of space.

The new set of rules is shown in Fig. 4. The algorithm that uses these rules is called the *optimised algorithm*. We use $\bowtie$ as a placeholder for either $\leq$ or $\geq$. The definition of *clash* is taken from the standard algorithm. We do not need a $\rightarrow_{\leq}$-rule.

At first glance, the $\rightarrow_{\geq}$-rule may appear to be complicated and therefor is explained in more detail: Like the standard $\rightarrow_{\geq}$-rule, it is applicable to a c.s. that contains the constraint $x \models \langle R\rangle_{\geq n}\phi$ if there are not enough witnesses for this constraint, i. e., if there are less than n R-successors y of x with $y \models \phi \in S$. The rule then adds a new witness y to S. Unlike the standard algorithm, the optimised algorithm also adds additional constraints of the form $y \models (\sim)\psi$ to S for each formula ψ appearing in a constraint of the form $x \models \langle R\rangle_{\bowtie n}\psi$. Since we have suspended the application of the $\rightarrow_{\geq}$-rule until no other rule applies to x, by this time S contains all constraints of the form $x \models \langle R\rangle_{\bowtie n}\psi$ it will ever

$\rightarrow_{\wedge}$-, $\rightarrow_{\vee}$-rule: see Fig. 1

$\rightarrow_{\geq}$-rule: if 1. $x \models \langle R\rangle_{\geq n}\phi \in S$, and
2. $\sharp R^S(x,\phi) < n$, and
3. neither the $\rightarrow_{\wedge}$- nor the $\rightarrow_{\vee}$-rule apply to a constraint for x

then $S \rightarrow_{\geq} S \cup \{Rxy, y \models \phi, y \models \chi_1, \ldots, y \models \chi_k\}$ where
$\{\psi_1, \ldots, \psi_k\} = \{\psi \mid x \models \langle R\rangle_{\bowtie m}\psi \in S\}$, $\chi_i \in \{\psi_i, \sim\psi_i\}$, and
y is a fresh variable.

Fig. 4. The optimised completion rules.

contain. This combines the effects of both the $\rightarrow_{\text{choose}}$- and the $\rightarrow_{\leq}$-rule of the standard algorithm.

4.1 Correctness of the Optimised Algorithm

To establish the correctness of the optimised algorithm, we will show its termination, soundness, and completeness.

To analyse the memory usage of the algorithm it is very helpful to view a c.s. as a graph: A c.s. S induces a labeled graph $G(S) = (N, E, \mathcal{L})$ with

- The set of nodes N is the set of variables appearing in S.
- The edges E are defined by $E := \{xy \mid Rxy \in S \text{ for some } R \in \mathcal{R}\}$.
- $\mathcal{L}$ labels nodes and edges in the following way:
  - For a node $x \in N$: $\mathcal{L}(x) := \{\phi \mid x \models \phi \in S\}$.
  - For an edge $xy \in E$: $\mathcal{L}(xy) := \{R \mid Rxy \in S\}$.

It is easy to show that the graph $G(S)$ for a c.s. S generated by the optimised algorithm from an initial c.s. $\{x_0 \models \phi\}$ is a tree with root x_0, and for each edge $xy \in E$, the label $\mathcal{L}(xy)$ is a singleton. Moreover, for each $x \in N$ it holds that $\mathcal{L}(x) \subseteq \mathit{clos}(\phi)$ where $\mathit{clos}(\phi)$ is the smallest set of formulae satisfying

- $\phi \in \mathit{clos}(\phi)$,
- if $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \psi_2 \in \mathit{clos}(\phi)$, then also $\psi_1, \psi_2 \in \mathit{clos}(\phi)$,
- if $\langle R \rangle_{\bowtie n}\psi \in \mathit{clos}(\phi)$, then also $\psi \in \mathit{clos}(\phi)$,
- if $\psi \in \mathit{clos}(\phi)$, then also $\sim\!\psi \in \mathit{clos}(\phi)$.

Without further proof we will us the fact that the number of elements of $\mathit{clos}(\phi)$ is bounded by $2 \times |\phi|$ where $|\phi|$ denotes the length of ϕ.

Termination. First, we will show that the optimised algorithm always terminates, i.e., each sequence of rule applications starting from a c.s. of the form $\{x_0 \models \phi\}$ is finite. The next lemma will also be of use when we will consider the complexity of the algorithm.

Lemma 1. *Let ϕ be a formula in NNF and S a c.s. that is generated by the optimised algorithm starting from $\{x_0 \models \phi\}$.*

- *The length of a path in $G(S)$ is limited by $|\phi|$.*
- *The out-degree of $G(S)$ is bounded by $|\mathit{clos}(\phi)| \times 2^{|\phi|}$.*

Proof. For a variable $x \in N$, we define $\ell(x)$ as the maximum depth of nested modalities in $\mathcal{L}(x)$. Obviously, $\ell(x_0) \leq |\phi|$ holds. Also, if $xy \in E$ then $\ell(x) > \ell(y)$. Hence each path $x_1, \ldots, x_k$ in $G(S)$ induces a sequence $\ell(x_1) > \cdots > \ell(x_k)$ of natural numbers. $G(S)$ is a tree with root x_0, hence the longest path in $G(S)$ starts with x_0 and its length is bounded by $|\phi|$.

Successors in $G(S)$ are only generated by the $\rightarrow_{\geq}$-rule. For a variable x this rule will generate at most n successors for each $\langle R \rangle_{\geq n}\psi \in \mathcal{L}(x)$. There are at most $|\mathit{clos}(\phi)|$ such formulae in $\mathcal{L}(x)$. Hence the out-degree of x is bounded by $|\mathit{clos}(\phi)| \times 2^{|\phi|}$, where $2^{|\phi|}$ is a limit for the biggest number that may appears in ϕ if binary coding is used. $\square$

Corollary 1 (Termination). *Any sequence of rule applications starting from a c.s. $S = \{x_0 \models \phi\}$ of the optimised algorithm is finite.*

Proof. The sequence of rules induces a sequence of trees. The depth and the out-degree of these trees is bounded in $|\phi|$ by Lemma 1. For each variable x the label $\mathcal{L}(x)$ is a subset of the finite set $\mathit{clos}(\phi)$. Each application of a rule either

– adds a constraint of the form $x \models \psi$ and hence adds an element to $\mathcal{L}(x)$, or
– adds fresh variables to S and hence adds additional nodes to the tree $G(S)$.

Since constraints are never deleted and variables are never identified, an infinite sequence of rule application must either lead to an arbitrary large number of nodes in the trees which contradicts their boundedness, or it leads to an infinite label of one of the nodes x which contradicts $\mathcal{L}(x) \subseteq \mathit{clos}(\phi)$. □

Soundness and Completeness. The following definition will be very helpful to establish soundness and completeness of the optimised algorithm:

Definition 4. *A c.s. S is called* satisfiable *iff there exists a Kripke structure $\mathfrak{M} = (W^{\mathfrak{M}}, \{R^{\mathfrak{M}} \mid R \in \mathcal{R}\}, V^{\mathfrak{M}})$ and a mapping $\alpha : \mathcal{V} \to W^{\mathfrak{M}}$ such that the following properties hold:*

1. *If y, z are distinct variables such that $Rxy, Rxz \in S$, then $\alpha(y) \neq \alpha(z)$.*
2. *If $x \models \psi \in S$ then $\mathfrak{M}, \alpha(x) \models \psi$.*
3. *If $Rxy \in S$ then $(\alpha(x), \alpha(y)) \in R^{\mathfrak{M}}$.*

In this case, $\mathfrak{M}, \alpha$ is called a model *of S.*

It easily follows from that definition, that a c.s. S that contains a clash can not be satisfiable and that the c.s. $\{x_0 \models \phi\}$ is satisfiable if and only if ϕ is satisfiable.

Lemma 2 (Local Correctness). *Let S, S' be c.s. generated by the optimised algorithm from a c.s. of the form $\{x_0 \models \phi\}$.*

1. *If S' is obtained from S by application of the (deterministic) $\to_\wedge$-rule, then S is satisfiable if and only if S' is satisfiable.*
2. *If S' is obtained from S by application of the (non-deterministic) $\to_\vee$- or $\to_\geq$-rule, then S is satisfiable if S' is satisfiable. Moreover, if S is satisfiable, then the rule can always be applied in such a way that it yields a c.s. S' that is satisfiable.*

Proof. $S \to S'$ for any rule $\to$ implies $S \subseteq S'$, hence each model of S' is also a model of S. Consequently, we must show only the other direction.

1. Let $\mathfrak{M}, \alpha$ be a model of S and let $x \models \psi_1 \wedge \psi_2$ be the constraint that triggers the application of the $\to_\wedge$-rule. The constraint $x \models \psi_1 \wedge \psi_2 \in S$ implies $\mathfrak{M}, \alpha(x) \models \psi_1 \wedge \psi_2$. This implies $\mathfrak{M}, \alpha(x) \models \psi_i$ for $i = 1, 2$. Hence $\mathfrak{M}, \alpha$ is also a model of $S' = S \cup \{x \models \psi_1, x \models \psi_2\}$.

2. Firstly, we consider the $\rightarrow_\vee$-rule. Let $\mathfrak{M}, \alpha$ be a model of S and let $x \models \psi_1 \vee \psi_2$ be the constraint that triggers the application of the $\rightarrow_\vee$-rule. $x \models \psi_1 \vee \psi_2 \in S$ implies $\mathfrak{M}, \alpha(x) \models \psi_1 \vee \psi_2$. This implies $\mathfrak{M}, \alpha(x) \models \psi_1$ or $\mathfrak{M}, \alpha(x) \models \psi_2$. Without loss of generality we may assume $\mathfrak{M}, \alpha(x) \models \psi_1$. The $\rightarrow_\vee$-rule may choose $\chi = \psi_1$, which implies $S' = S \cup \{x \models \psi_1\}$ and hence $\mathfrak{M}, \alpha$ is a model for S'.
   Secondly, we consider the $\rightarrow_\geq$-rule. Again let $\mathfrak{M}, \alpha$ be a model of S and let $x \models \langle R \rangle_{\geq n} \phi$ be the constraint that triggers the application of the $\rightarrow_\geq$-rule. Since the $\rightarrow_\geq$-rule is applicable, we have $\sharp R^S(x, \phi) < n$. We claim that there is a $w \in W^{\mathfrak{M}}$ with
   $$(\alpha(x), w) \in R^{\mathfrak{M}}, \mathfrak{M}, w \models \phi, \text{ and } w \notin \{\alpha(y) \mid Rxy \in S\}. \qquad (*)$$
   Before we prove this claim, we show how it can be used to finish the proof. The world w is used to "select" a choice of the $\rightarrow_\geq$-rule that preserves satisfiability: Let $\{\psi_1, \ldots, \psi_n\}$ be an enumeration of the set $\{\psi \mid x \models \langle R \rangle_{\bowtie n} \psi \in S\}$. We set
   $$S' = S \cup \{Rxy, y \models \phi\} \cup \{y \models \psi_i \mid \mathfrak{M}, w \models \psi_i\} \cup \{y \models {\sim}\psi_i \mid \mathfrak{M}, w \not\models \psi_i\}.$$
   Obviously, $\mathfrak{M}, \alpha[y \mapsto w]$ is a model for S' (since y is a fresh variable and w satisfies $(*)$), and S' is a possible result of the application of the $\rightarrow_\geq$-rule to S.

We will now come back to the claim. It is obvious that there is a w with $(\alpha(x), w) \in R^{\mathfrak{M}}$ and $\mathfrak{M}, w \models \phi$ that is not contained in $\{\alpha(y) \mid Rxy, y \models \phi \in S\}$, because $\sharp R^{\mathfrak{M}}(x, \phi) \geq n > \sharp R^S(x, \phi)$. Yet w might appear as the image of an element y' such that $Rxy' \in S$ but $y' \models \phi \notin S$.

Now, $Rxy' \in S$ and $y' \models \phi \notin S$ implies $y' \models {\sim}\phi \in S$. This is due to the fact that the constraint Rxy' must have been generated by an application of the $\rightarrow_\geq$-rule because it has not been an element of the initial c.s. The application of this rule was suspended until neither the $\rightarrow_\wedge$- nor the $\rightarrow_\vee$-rule are applicable to x. Hence, if $x \models \langle R \rangle_{\geq n} \phi$ is an element of S by now, then it has already been in S when the $\rightarrow_\geq$-rule that generated y', was applied. The $\rightarrow_\geq$-rule guarantees that either $y' \models \phi$ or $y' \models {\sim}\phi$ is added to S. Hence $y' \models {\sim}\phi \in S$. This is a contradiction to $\alpha(y') = w$ because under the assumption that $\mathfrak{M}, \alpha$ is a model of S this would imply $\mathfrak{M}, w \models {\sim}\phi$ while we initially assumed $\mathfrak{M}, w \models \phi$. □

From the local completeness of the algorithm we can immediately derive the global completeness of the algorithm:

Lemma 3 (Completeness). *If $\phi \in \mathrm{SAT}(\mathbf{Gr}(\mathbf{K}_{\mathcal{R}}))$ in NNF, then there is a sequence of applications of the optimised rules starting with $S = \{x_0 \models \phi\}$ that results in a complete and clash-free c.s.*

Proof. The satisfiability of ϕ implies that also $\{x_0 \models \phi\}$ is satisfiable. By Lemma 2 there is a sequence of applications of the optimised rules which preserves the satisfiability of the c.s. By Lemma 1 any sequence of applications must be finite. No generated c.s. (including the last one) may contain a clash because this would make them unsatisfiable. □

Note that since we have made no assumption about the order in which the rules are applied (with the exception that is stated in the conditions of the $\rightarrow_{\geq}$-rule), the selection of the constraints to apply a rule to as well as the selection which rule to apply is "don't-care" non-deterministic, i.e., if a formula is satisfiable, then this can be proved by an arbitrary sequence of rule applications. Without this property, the resulting algorithm certainly would be useless for practical applications, because any deterministic implementation would have to use backtracking on the selection of constraints and rules.

Lemma 4 (Soundness). *Let ϕ be a $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$-formula in NNF. If there is a sequence of applications of the optimised rules starting with the c.s. $\{x_0 \models \phi\}$ that results in a complete and clash-free c.s., then $\phi \in \mathrm{SAT}(\mathbf{Gr}(\mathbf{K}_{\mathcal{R}}))$.*

Proof. Let S be a complete and clash-free c.s. generated by applications of the optimised rules. We will show that the canonical model $\mathfrak{M}_S$ together with the identity function is a model for S. Since S was generated from $\{x_0 \models \phi\}$ and the rules do not remove constraints from the c.s., $x_0 \models \phi \in S$. Thus $\mathfrak{M}_S$ is also a model for ϕ with $\mathfrak{M}_S, x_0 \models \phi$.

By construction of $\mathfrak{M}_S$, Property 1 and 3 of Definition 4 are trivially satisfied. It remains to show that $x \models \psi \in S$ implies $\mathfrak{M}, x \models \psi$, which we will show by induction on the norm $\|\cdot\|$ of a formula ψ. The norm $\|\psi\|$ for formulae in NNF is inductively defined by:

$$\begin{array}{lll} \|p\| & := \|\neg p\| & := 0 \quad \text{for } p \in \mathcal{P} \\ \|\psi_1 \wedge \psi_2\| & := \|\psi_1 \vee \psi_2\| & := 1 + \|\psi_1\| + \|\psi_2\| \\ \|\langle R \rangle_{\bowtie n} \psi\| & & := 1 + \|\psi\| \end{array}$$

This definition is chosen such that it satisfies $\|\psi\| = \|{\sim}\psi\|$ for every formula ψ.

- The first base case is $\psi = p$ for $p \in \mathcal{P}$. $x \models p \in S$ implies $x \in V^{\mathfrak{M}_S}(p)$ and hence $\mathfrak{M}_S, x \models p$. The second base case is $x \models \neg p \in S$. Since S is clash-free, this implies $x \models p \notin S$ and hence $x \notin V^{\mathfrak{M}_S}(p)$. This implies $\mathfrak{M}_S, x \models \neg p$.
- $x \models \psi_1 \wedge \psi_2 \in S$ implies $x \models \psi_1, x \models \psi_2 \in S$. By induction, we have $\mathfrak{M}_S, x \models \psi_1$ and $\mathfrak{M}_S, x \models \psi_2$ holds and hence $\mathfrak{M}_S, x \models \psi_1 \wedge \psi_2$. The case $x \models \psi_1 \vee \psi_2 \in S$ can be handled analogously.
- $x \models \langle R \rangle_{\geq n} \psi \in S$ implies $\sharp R^S(x, \psi) \geq n$ because otherwise the $\rightarrow_{\geq}$-rule would be applicable and S would not be complete. By induction, we have $\mathfrak{M}_S, y \models \psi$ for each y with $y \models \psi \in S$. Hence $\sharp R^{\mathfrak{M}_S}(x, \psi) \geq n$ and thus $\mathfrak{M}_S, x \models \langle R \rangle_{\geq n} \psi$.
- $x \models \langle R \rangle_{\leq n} \psi \in S$ implies $\sharp R^S(x, \psi) \leq n$ because S is clash-free. Hence it is sufficient to show that $\sharp R^{\mathfrak{M}_S}(x, \psi) \leq \sharp R^S(x, \psi)$ holds. On the contrary, assume $\sharp R^{\mathfrak{M}_S}(x, \psi) > \sharp R^S(x, \psi)$ holds. Then there is a variable y such that $Rxy \in S$ and $\mathfrak{M}_S, y \models \psi$ while $y \models \psi \notin S$. For each variable y with $Rxy \in S$ either $y \models \psi \in S$ or $y \models {\sim}\psi \in S$. This implies $y \models {\sim}\psi \in S$ and, by the induction hypothesis, $\mathfrak{M}_S, y \models {\sim}\psi$ holds which is a contradiction. □

The following theorem is an immediate consequence of Lemma 1, 3, and 4:

Corollary 2. *The optimised algorithm is a non-deterministic decision procedure for* $\mathrm{SAT}(\mathbf{Gr}(\mathbf{K}_{\mathcal{R}}))$.

4.2 Complexity of the Optimised Algorithm

The optimised algorithm will enable us to prove Theorem 2. We will give a proof by sketching an implementation of this algorithm that runs in polynomial space.

Lemma 5. *The optimised algorithm can be implemented in* PSPACE

Proof. Let ϕ be the $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$-formula to be tested for satisfiability. We can assume ϕ to be in NNF because the transformation of a formula to NNF can be performed in linear time and space.

The key idea for the PSPACE implementation is the *trace technique*[SSS91], i.e., it is sufficient to keep only a single path (a trace) of $G(S)$ in memory at a given stage if the c.s. is generated in a depth-first manner. This has already been the key to a PSPACE upper bound for $\mathbf{K}_{\mathcal{R}}$ and $\mathcal{ALC}$ in [Lad77,SSS91,HM92]. To do this we need to store the values for $\sharp R^S(x, \psi)$ for each variable x in the path, each R which appears in $clos(\phi)$ and each $\psi \in clos(\phi)$. By storing these values in binary form, we are able to keep information *about* exponentially many successors in memory while storing only a single path at a given stage.

Consider the algorithm in Fig. 5, where $\mathcal{R}_\phi$ denotes the set of relation names that appear in $clos(\phi)$. It re-uses the space needed to check the satisfiability of a successor y of x once the existence of a complete and clash-free "subtree" for the constraints on y has been established. This is admissible since the optimised rules will never modify change this subtree once is it completed. Neither do constraints in this subtree have an influence on the completeness or the existence of a clash in the rest of the tree, with the exception that constraints of the form $y \models \psi$ for R-successors y of x contribute to the value of $\sharp R^S(x, \psi)$. These numbers play a role both in the definition of a clash and for the applicability of the $\rightarrow_{\geq}$-rule. Hence, in order to re-use the space occupied by the subtree for y, it is necessary and sufficient to store these numbers.

Let us examine the space usage of this algorithm. Let $n = |\phi|$. The algorithm is designed to keep only a single path of $G(S)$ in memory at a given stage. For each variable x on a path, constraints of the form $x \models \psi$ have to be stored for formulae $\psi \in clos(\phi)$. The size of $clos(\phi)$ is bounded by $2n$ and hence the constraints for a single variable can be stored in $\mathcal{O}(n)$ bits. For each variable, there are at most $|\mathcal{R}_\phi| \times |clos(\phi)| = \mathcal{O}(n^2)$ counters to be stored. The numbers to be stored in these counters do not exceed the out-degree of x, which, by Lemma 1, is bounded by $|clos(\phi)| \times 2^{|\phi|}$. Hence each counter can be stored using $\mathcal{O}(n^2)$ bits when binary coding is used to represent the counters, and all counters for a single variable require $\mathcal{O}(n^4)$ bits. Due to Lemma 1, the length of a path is limited by n, which yields an overall memory consumption of $\mathcal{O}(n^5 + n^2)$. $\square$

Theorem 2 now is a simple Corollary from the PSPACE-hardness of $\mathbf{K}_{\mathcal{R}}$, Lemma 5, and Savitch's Theorem [Sav70].

5 Conclusion

We have shown that by employing a space efficient tableaux algorithm satisfiability of $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ can be decided in PSPACE, which is an optimal result with

```
$\mathbf{Gr}(\mathbf{K}_{\mathcal{R}}) - \mathrm{SAT}(\phi) :=$ sat$(x_0, \{x_0 \models \phi\})$
sat$(x, S)$:
    allocate counters $\sharp R^S(x, \psi) := 0$ for all $R \in \mathcal{R}_\phi$ and $\psi \in clos(\phi)$.
    while (the $\rightarrow_\wedge$- or the $\rightarrow_\vee$-rule can be applied) and ($S$ is clash-free) do
        apply the $\rightarrow_\wedge$- or the $\rightarrow_\vee$-rule to $S$.
    od
    if $S$ contains a clash then return "not satisfiable".
    while (the $\rightarrow_\geq$-rule applies to $x$ in $S$) do
        $S_{new} := \{Rxy, y \models \phi', y \models \chi_1, \ldots, y \models \chi_k\}$
        where
            $y$ is a fresh variable,
            $x \models \langle R \rangle_{\geq n} \phi'$ triggers an application of the $\rightarrow_\geq$-rule,
            $\{\psi_1, \ldots, \psi_k\} = \{\psi \mid x \models \langle R \rangle_{\bowtie n} \psi \in S\}$, and
            $\chi_i$ is chosen non-deterministically from $\{\psi_i, \sim\psi_i\}$
        for each $y \models \psi \in S_{new}$ do increase $\sharp R^S(x, \psi)$
        if $x \models \langle R \rangle_{\leq m} \psi \in S$ and $\sharp R^S(x, \psi) > m$ then return "not satisfiable".
        if sat$(y, S_{new})$ = "not satisfiable" then return "not satisfiable"
    od
    remove the counters for $x$ from memory.
    return "satisfiable"
```

Fig. 5. A non-deterministic PSpace decision procedure for SAT($\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$).

respect to worst-case complexity. It is possible to obtain an analogous result for the DL $\mathcal{ALCQR}$ by applying similar techniques. $\mathcal{ALCQR}$, which strictly extends the expressivity of $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ by allowing for relation intersection $R_1 \cap \cdots \cap R_m$ in the modalities, contains the DL $\mathcal{ALCNR}$ for which the upper complexity bound with binary coding had also been an open problem [DLNN97]. While the algorithm presented certainly is only optimal from the viewpoint of worst-case complexity, it is relatively simple and will serve as the starting-point for a number of optimisations leading to more practical implementations. It also serves as a tool to establish the upper complexity bound of the problem and thus shows that tableaux based reasoning for $\mathbf{Gr}(\mathbf{K}_{\mathcal{R}})$ can be done with optimum worst-case complexity. This establishes a kind of "theoretical benchmark" that all algorithmic approaches can be measured with.

Acknowledgments. I would like to thank Franz Baader and Ulrike Sattler for valuable comments and suggestions.

References

AvBN98. H. Andréka, J. van Benthem, and I. Németi Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

BBH96. F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.

CLN94. D. Calvanese, M. Lenzerini, and D. Nardi. A Unified Framework for Class Based Representation Formalisms. *Proc. of KR-94*, 1994.

dHR95. W. Van der Hoek and M. De Rijke. Counting objects. *Journal of Logic and Computation*, 5(3):325–345, June 1995.

DLNN97. F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134(1):1–58, 10 April 1997.

Fin72. K. Fine. In so many possible worlds. *Notre Dame Journal of Formal Logic*, 13:516–520, 1972.

GS96. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures—the case study of modal K. *Proc. of CADE-13*, LNCS 1104. Springer, 1996.

HB91. B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proc. of KR-91*, pages 335–346, Boston (USA), 1991.

HM92. J. Y. Halpern and Y. Moses. A guide to completeness and complexity for model logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, April 1992.

HS97. U. Hustadt and R. A. Schmidt. On evaluating decision procedures for modal logic. In *Proc. of IJCAI-97)*, volume 1, pages 202–207, 1997.

Lad77. R. E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480, September 1977.

OS97. H. J. Ohlbach and R. A. Schmidt. Functional translation and second-order frame properties of modal logics. *Journal of Logic and Computation*, 7(5):581–603, October 1997.

OSH96. H. J. Ohlbach, R. A. Schmidt, and U. Hustadt. Translating graded modalities into predicate logic. In H. Wansing, editor, *Proof Theory of Modal Logic*, volume 2 of *Applied Logic Series*, pages 253–291. Kluwer, 1996.

Sav70. W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, April 1970.

Sch91. K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI-91*, pages 466–471, 1991.

Sch97. R. A. Schmidt. Resolution is a decision procedure for many propositional modal logics: Extended abstract. In M. Kracht, M. de Rijke, H. Wansing, and M. Zakharyaschev, editors, *Advances in Modal Logic '96*. CLSI Publications, 1997.

SSS91. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

Solvability of Context Equations with Two Context Variables is Decidable

Manfred Schmidt-Schauß[1] and Klaus U. Schulz[2]*

[1] Fachbereich Informatik, J.-W.-Goethe-Universität, Postfach 11 19 32
D-60054 Frankfurt, Germany
schauss@ki.informatik.uni-frankfurt.de

[2] CIS, University of Munich, Oettingenstr 67, D-80538 München, Germany
schulz@cis.uni-muenchen.de

Abstract. Context unification is a natural variant of second order unification that represents a generalization of word unification at the same time. While second order unification is wellknown to be undecidable and word unification is decidable it is currently open if solvability of context equations is decidable. We show that solvability of systems of context equations with two context variables is decidable. The context variables may have an arbitrary number of occurrences, and the equations may contain an arbitrary number of individual variables as well. The result holds under the assumption that the first-order background signature is finite.

1 Introduction

Context unification studies solvability of equations that describe composition of ground terms by means of ground subcontexts and ground subterms. A ground context is a ground term with exactly one occurrence of a special constant that represents a missing argument, or a lambda-bound variable. The main result of this paper is the following

Theorem 1 (Main Theorem). *Solvability of finite systems of context equations with two context variables is decidable.*

Both context variables may have an arbitrary number of occurrences, and the equations may contain an arbitrary number of individual variables as well. The result holds under the assumption that the first-order background signature is finite. The paper provides a partial solution to the *Context Unification Problem* that has recently attracted considerable attention [1,6,22,10,11,16]: *Is solvability of arbitrary context equations decidable?* The interest in this problem relies on its close connection to several other well-studied decision problems.

Context unification represents a natural variant of second order unification, which is undecidable ([4,2,7]). The main difference is that in second order unification second order variables are instantiated by terms with an arbitrary number

* partially funded by the ESPRIT Working Group CCL II, EP: 22457.

of lambda-bound variables (dependent on their arity) and each such variable may have an arbitrary number $n \geq 0$ of occurrences in the substitution instance. In context unification context variables are instantiated by terms with just one lambda-bound variable that has exactly one occurrence. A recent result [15] shows that second order unification becomes decidable if an upper bound on the number of occurrences of a given bound variable in the substitution term is fixed. If context unification would turn out to be decidable, the latter result would be a simple consequence. It has been shown that second order unification is undecidable even for problems with one second order variable only [3][1]. Hence our result shows that there is at least some significant difference between context unification and second order unification.

Context unification can also be considered as a generalization of word unification [8,18,19,5]. Decidability of word unification had been an open problem for many years before G. S. Makanin [8] could show that solvability of arbitrary word equations is decidable. In [18] Makanin's result was generalized to word equations with so-called regular constraints. This result will be used here to prove the Main Theorem.

Context unification is used as a formalism for semantic analysis of natural language utterances [10,11]. It also occurs in the context of distributive unification [14] and completion of rewrite systems with membership constraints [1]. J. Levy [6] has shown that solvability of context unification problems where each variable occurs at most twice is decidable. In [13] there is an algorithm and a sketch of a proof showing that solvability of so-called stratified context unification problems is decidable; a complete proof for a restricted signature is published in [14]. In [16] the authors have given an upper bound on the so-called exponent of periodicity of a minimal solution of a context equation. J. Niehren, M. Pinkal and P. Ruhrberg [10] showed that context unification and so-called "equality up-to constraints" are equally expressive and that one-step rewriting constraints can be expressed by stratified context unification problems. Recently, it was noticed [12] that the converse is also true, which shows that stratified context unification and one-step rewriting constraints are interreducible. It was also noticed in [10] that the first-order theory of context unification is undecidable, using the fact that the first-order theory of one-step rewriting is undecidable [20,21,9]. This result was improved by S. Vorobyov [22] who showed that the $\forall\exists^8$ equational theory of context unification is co-recursively enumerable hard.

The proof of Theorem 1 uses a series of four non-deterministic translation steps, called (Transl1)–(Transl4). Using (Transl1) a given context equation is non-deterministically transformed into a so-called generalized context problem (Section 4). Such a problem gives a partial description of a ground term that represents a (hypothetical) solution of the given equation.

The intermediate steps (Transl2) and (Transl3) are strongly adapted to the needs of the final translation (Transl4), for this reason we first describe the latter procedure in Section 5. Using (Transl4), generalized context problems

[1] The result was first presented by J. Levy and M. Veanes at CSL'98.

of a particular form are translated into systems of word equations with regular constraints. The translation does not depend on the number of context variables.

Translations 2 and 3 are used to bring generalized context problems into a form acceptable for (Transl4). The first step, (Transl2), is rather simple and works for an arbitrary number of context variables. (Transl3), to be described in Section 6, is the real source of difficulties. Here termination can only be guaranteed for input problems with two context variables only.

In Section 7 we combine the results for the single translation steps and prove Theorem 1. Here we restrict considerations to single context equations. Due to space limitations, other proofs have to be omitted here. All proofs (as well as some formalities that have been replaced in favour of intuitive descriptions, following suggestions of the referees) can be found in [17].

2 Formal Preliminaries

Throughout this paper, Σ denotes a finite signature of function symbols, having at least one constant. A special constant $\Omega \notin \Sigma$ will be used. $\Sigma^{\Omega} := \Sigma \cup \{\Omega\}$ denotes the extended signature. $\mathcal{V}$ denotes an infinite set of *context variables* $X, Y, Z, \ldots$. We shall also use individual variables $x, y, z, \ldots$, and $\mathcal{X}$ denotes the set of individual variables.

We first give syntax and semantics of context unification. To begin with, *ground terms* over Σ and (occurrences of) *subterms* are defined as usual.

Definition 1. *A* ground context *is a ground Σ^{Ω}-term t that has exactly one occurrence of the constant Ω. Ω is called the* empty ground context.

Given a ground context s and a ground term/context t we write st for the ground term/context that is obtained from s when we replace the occurrence of Ω in s by t. Note that this form of composition is associative.

Definition 2. Context terms *over Σ, $\mathcal{X}$ and $\mathcal{V}$ are defined as follows:*

- *each individual variable $x \in \mathcal{X}$ is a context term,*
- *if $t_1, \ldots, t_n$ are context terms and $f \in \Sigma$ is n-ary, then $f(t_1, \ldots, t_n)$ is a context term ($n \geq 0$),*
- *if t is a context term and $X \in \mathcal{V}$, then $X(t)$ is a context term.*

A context equation *is an equation of the form $s = t$ where s and t are context terms.*

Definition 3. *A* substitution *is a mapping S that assigns a ground context $S(X)$ to each $X \in \mathcal{V}$, and a ground term $S(x)$ to each $x \in \mathcal{X}$. The mapping S is extended to arbitrary context terms as follows*

- *$S(f(t_1, \ldots, t_n)) := f(S(t_1), \ldots, S(t_n))$ for n-ary $f \in \Sigma$ ($n \geq 0$),*
- *$S(X(t)) := S(X)S(t)$ for $X \in \mathcal{V}$.*

A substitution S is a solution *of the context equation $s = t$ if $S(s) = S(t)$.*

Example 1. Let $\Sigma := \{f, g, a\}$ where f is binary, g is unary, and a is a constant. The context equation $X(X(a)) = f(Y(f(Y(a), Z(a))), x)$ is solved by the substitution $X \mapsto f(g(\Omega), g(g(a)))$, $Y \mapsto g(\Omega)$, $Z \mapsto g(g(\Omega)), x \mapsto g(g(a))$ since under this substitution both sides are mapped to $f(g(f(g(a), g(g(a)))), g(g(a)))$.

A solution S of the context equation $s = t$ is *positive* if S maps each context variable appearing in $s = t$ to a *non-empty* ground context. Clearly, in order to decide solvability of a given context equation $s = t$ it suffices to have a procedure for deciding positive solvability: we may simply guess which context variables are instantiated by the empty ground context and instantiate these variables by Ω in the equation $s = t$. *In the sequel, with a solution of a context equation we always mean a positive solution.*

The ground context s is a *subcontext* of the ground term t if there exists a ground term t_1 and a ground context r such that $t = rst_1$. Note that in this situation the root of the given occurrence of t_1 marks the position of the (filled) "hole" Ω of s.

Definition 4. *Let t be a ground context. The* main path *of t is the path from the root to Ω. The* side area *of t is the set of all nodes that do not belong to the main path.*

We shall also talk about the main path/side area of a given occurrence of a ground context as a subcontext of a ground term.

The following definition yields the basis for the correspondence between ground contexts and words that is used at the final translation step.

Definition 5. *A ground context C is a* letter *if the hole Ω of C is in depth 1. For each ground context C with hole in depth k there exists a unique sequence of letters $C_1, \ldots, C_k$ such that $C = C_1 \cdots C_k$. The elements $C_1, \ldots, C_k$ are called the* letters *of C.*

Definition 6. *A* tree domain *is a finite, ordered and unlabeled tree N. The notions of* nodes, descendants, ancestors, children, leaves, inner nodes *are defined as usual. A node of N is* branching *if it has at least two distinct children in N. Two nodes η_1 and η_2 are* incompatible *if $\eta_1 \neq \eta_2$ and neither η_1 is an ancestor of η_2 nor vice versa. A* subtree domain *of a tree domain N is a subset N' of N that is a tree domain.*

Definition 7. *Let N be a tree domain. A* field *of N is a sequence of nodes $\varphi = (\eta_0, \ldots, \eta_k)$ ($k \geq 1$) such that η_{i+1} is a child of η_i, for $0 \leq i \leq k-1$. Node η_0 (η_k) is the* initial *(resp.* final*) node of the field, the number k is the* length *of the field. The* side area *(or set of* side nodes*) of $(\eta_0, \ldots, \eta_k)$ is the set of all nodes of N that are descendants of one of the nodes $\eta_0, \ldots, \eta_{k-1}$, but neither in $\{\eta_0, \ldots, \eta_k\}$ nor in the set of descendants of η_k. Fields of length 1 are called* atomic. *Two fields of N are* branching *if they have a common node and if the final nodes are incompatible. The maximal common prefix of the final nodes is called the* branching point *of the two fields. Let η_1 and η_2 be incompatible nodes. The maximal common prefix (i.e., the closest common ancestor) of η_1 and η_2 is denoted $MCP_N(\eta_1, \eta_2)$.*

Definition 8. *A* labeled tree domain *is a pair* (N, Lab) *where* N *is a tree domain and* $Lab : N \to \Sigma$ *is a partial function such that* $Lab(\eta) = f \in \Sigma$ *implies that* η *has exactly* k *children, where* f *is* k*-ary.*

Note that each ground term in a natural way represents a labeled tree domain with a total labeling function. In the sequel, we do not distinguish between these two notions.

Definition 9. *If* $\varphi = (\eta_0, \ldots, \eta_k)$ *is a field of the ground term* t*, then* φ *together with its side area defines a unique non-empty ground subcontext* s *of* t*. The root of* s *is given by* η_0*, and* η_k *marks the position of the hole* Ω*. The subcontext* s *is called the* ground subcontext of t with main path φ*.*

If $F : N_1 \to N_2$ is a mapping between two labeled tree domains (N_1, Lab_1) and (N_2, Lab_2), we say that F *respects branching points* if $F(\mathrm{MCP}_{N_1}(\eta_1, \eta_2)) = \mathrm{MCP}_{N_2}(F(\eta_1), F(\eta_2))$ for all incompatible nodes η_1, η_2 of N_1. We say that F *respects children relationship for labeled nodes* if each child of a labeled node $\eta \in N_1$ is mapped to a child of $F(\eta)$ under F. The *preorder* relationship on nodes is defined as usual.

Definition 10. *Let* (N_1, Lab_1) *and* (N_2, Lab_2) *be labeled tree domains. An injective mapping* $F : N_1 \to N_2$ *is a* labeled tree embedding *if* F *respects root,* Σ*-labels, preorder relationship, branching points, and children relationship for labeled nodes.*

A labeled tree embedding may map an unlabeled node to a labeled node. If $\varphi = (\eta_0, \ldots, \eta_k)$ is a field of N_1 we write $F(\varphi)$ or $F(\eta_0, \ldots, \eta_k)$ for the field of N_2 with initial (resp. final) node $F(\eta_0)$ (resp. $F(\eta_k)$).

3 Generalized Context Problems

The central data structure for the intermediate translation steps is the following.

Definition 11. *A* generalized context problem *over the signature* Σ *is a tuple* $T = \langle N, Lab, CB, Field, IB, Node \rangle$ *where*

1. (N, Lab) *is a labeled tree domain.*
2. CB *is a finite set of so-called* context bases*. Each context base has a unique type, which is a context variable. Context bases are written in the form* cb*, or in the form* $X^{(i)}$ *where* X *is the type and* $i \in \mathbb{N}$.
3. *Field is a function that assigns to each context base* $cb \in CB$ *a field* $Field(cb)$ *of* N*.*
4. IB *is a finite set of so-called* individual bases*. Each individual base has a unique type, which is an individual variable. Individual bases are written in the form* ib*, or in the form* $x^{(i)}$ *where* x *is the type and* $i \in \mathbb{N}$*.*
5. $Node : IB \to N$ *is a total function.*

The following conditions have to be satisfied:

6. *each child of an unlabeled node of T is a non-initial node of the field of a context base of T,*
7. *each leaf η of N is either labeled with an individual constant in Σ or there exists an individual base $ib \in IB$ with $Node(ib) = \eta$.*

T is called first-order *if CB and $Field$ are empty.*

See Example 2 for a graphical representation of a generalized context problem. If T is first-order, then there are no fields of context bases and thus, by Condition 6, each inner node of T is labeled. In this case T can be considered as a first-order term where in addition nodes may be decorated with individual variables. More general, given the notion of a solution as introduced below, Condition 6 ensures that the values of the bases uniquely determine the solution.

Since each context base has a unique field we may also refer to the final (resp. initial) node and similarly to the side area of a context base. Two context bases are *branching* if their fields are branching. The branching point of the fields is also called the branching point of the two context bases.

Definition 12. *Let $T = \langle N, Lab, CB, Field, IB, Node\rangle$ be a generalized context problem. A* solution *of T is a pair (t, S) where t is a ground term and S is a labeled tree embedding from N to the set of nodes of t such that the following conditions hold:*

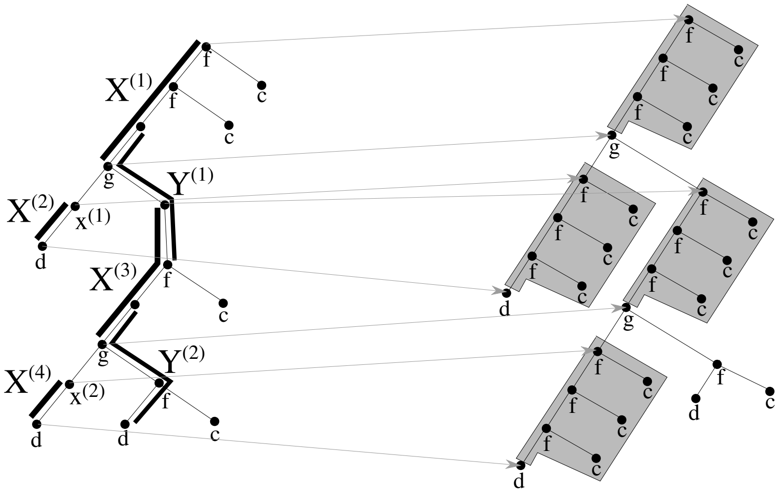
1. *for all context bases $X^{(i)}$ and $X^{(j)}$ of the same type X of T the ground subcontexts of t with main paths $S(Field(X^{(i)}))$ and $S(Field(X^{(j)}))$ respectively are identical, and*
2. *for all individual bases $x^{(i)}$ and $x^{(j)}$ of the same type x of T the ground subterms of t with roots $S(Node(x^{(i)}))$ and $S(Node(x^{(j)}))$ respectively are identical.*

If (t, S) is a solution of T and η is a node of T we write $\hat{S}(\eta)$ for the subterm of t with root $S(\eta)$. If φ is a field of T we write $\hat{S}(\varphi)$ for the ground subcontext of t with main path $S(\varphi)$. If $X^{(i)}$ is a context base of T we write $\hat{S}(X^{(i)})$ for $\hat{S}(Field(X^{(i)}))$. Since all context bases of type X are mapped to the same ground context we also write $\hat{S}(X)$ instead of $\hat{S}(X^{(i)})$. Similary, if $x^{(i)}$ is an individual base of T we write $\hat{S}(x^{(i)})$ for the ground subterm $\hat{S}(Node(x^{(i)}))$ and we write $\hat{S}(x)$ instead of $\hat{S}(x^{(i)})$.

Definition 13. *Let $T = \langle N, Lab, CB, Field, IB, Node\rangle$ be a generalized context problem. An atomic subfield (η, η') of the field of a base $cb \in CB$, with labeled node η, is called a* letter description *of T (or of cb) with main node η and label $Lab(\eta)$. If η' is the i-th child of η, then (η, η') has* direction *i.*

We use symbols L, L_1 etc. to denote letter descriptions. Note that if (t, S) is a solution of T and L is a letter description of the context base cb of T, then the ground subcontext $\hat{S}(L)$ is a subletter of $\hat{S}(cb)$.

Example 2. The following figure represents a generalized context problem T (left-hand side) and the solution term t (right-hand side) of a solution (t, S). Bold lines represent the *Field*-function. Grey areas on the right-hand side represent the ground contexts $\hat{S}(X) = \hat{S}(X^{(1)}) = \cdots = \hat{S}(X^{(4)})$. T has six letter descriptions, see Example 4.



4 Translation into Generalized Context Problems

We define the notion of a superposition of two generalized context problems and use it for translating context equations into generalized context problems.

Definition 14. *Let $T_1 = \langle N_1, Lab_1, CB_1, Field_1, IB_1, Node_1\rangle$ and $T = \langle N, Lab, CB, Field, IB, Node\rangle$ be generalized context problems. An* embedding *of T_1 in T is a labeled tree embedding $F : N_1 \to N$ such that*

1. *for each context base $X^{(i)} \in CB_1$ there exists a context base $X^{(j)} \in CB$ of the same type such that $F(Field_1(X^{(i)})) = Field(X^{(j)})$,*
2. *each node of an individual base of type x of T_1 is mapped to the node of an individual base of the same type x of T under F, for all $x \in \mathcal{X}$.*

Definition 15. *Let $n \geq 2$. The generalized context problem T is a* superposition *of the generalized context problems $T_1, \ldots, T_n$ iff for $i = 1, \ldots, n$ there are embeddings F_i of T_i in T such that the following conditions hold:*

1. *For each context base $X^{(r)}$ of T there exists $i \in \{1, \ldots, n\}$ and a context base $X^{(s)}$ of the same type of T_i such that $Field(X^{(r)}) = F_i(Field_i(X^{(s)}))$,*
2. *For each individual base $x^{(r)}$ of T there exists $i \in \{1, \ldots, n\}$ and an individual base $x^{(s)}$ of the same type of T_i such that $Node(x^{(r)}) = F_i(Node_i(x^{(s)}))$,*
3. *For each node η of T either there exists $i \in \{1, \ldots, n\}$ and a node η' of T_i such that $\eta = F_i(\eta')$, or there are two distinct indices $i, j \in \{1, \ldots, n\}$ and bases $cb_1 \in CB_i$ and $cb_2 \in CB_j$ such that the images of cb_1 and cb_2 in T branch at η.*

4. *Modulo renaming of superscripts, the set of context (individual) bases of* T *is the union of the sets of context (individual) bases of the problems* $T_1, \ldots, T_n$.

Lemma 1. *For* $i = 1, \ldots, n$, *let* $T_i = \langle N_i, Lab_i, CB_i, Field_i, IB_i, Node_i \rangle$ *be a generalized context problem. Then the set of all superpositions of* $T_1, \ldots, T_n$ *is (modulo renaming of nodes/superscripts) finite. If, for some ground term* t, *each* T_i $(1 \leq i \leq n)$ *has a solution of the form* (t, S_i) *where the embeddings* S_i *assign the same ground context (resp. term) to context (individual) bases of the same type, then there exists a superposition* T *of* $T_1, \ldots, T_n$ *with a solution of the form* (t, S).

Lemma 2 (Transl1). *For each context equation* E *it is possible to compute a finite set* $\mathcal{T}$ *of generalized context problems such that* E *has a solution iff some* $T \in \mathcal{T}$ *has a solution. If* E *has occurrences of* k *context variables only, the same holds for the problems in* $\mathcal{T}$.

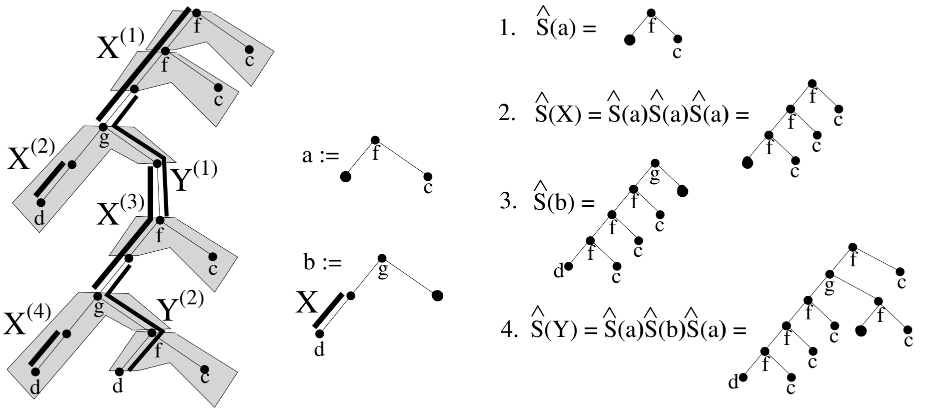
Proof (sketch). Consider a context equation $t_1 \doteq t_2$. Each of the context terms t_i can be considered as a trivially solvable generalized context problem T_i. The set $\mathcal{T}$ of all superpositons of T_1 and T_2 has the required properties. □

Example 3. Superpositioning the problems associated with the context terms $X(g(X(d), X(g(X(d), f(d, c)))))$ and $f(f(Y(f(Y(d), c)), c), c)$ we obtain (among other superpositions) the generalized context problem T depicted in Example 2.

5 Translation of Generalized Context Problems into Word Equations with Regular Constraints

We just illustrate this translation with a continuation of Example 2. All formal details of the translation and the correctness proof can be found in [17].

Example 4. The generalized context problem T given in Example 2 has six letter descriptions, highlighted on the left-hand side of the following figure. It is simple to see that under any solution all occurrences of letter descriptions of type a (cf. figure) are are mapped to the same ground context, the same holds for all letter descriptions of type b.



We introduce three auxiliary context bases: $U^{(1)}$ will mark the overlap of $X^{(1)}$ and $Y^{(1)}$, $V^{(1)}$ the overlap of $Y^{(1)}$ and $X^{(3)}$, and $W^{(1)}$ the overlap of $X^{(3)}$ and $Y^{(2)}$. The context to be substituted for $X^{(1)}$ has the form aaU, the context for $X^{(3)}$ the form VaW, no special form is needed for $X^{(2)}$ and $X^{(4)}$. Since all context bases $X^{(i)}$ have to be mapped to the same ground context we obtain $X = aaU = VaW$. In the same way we obtain the equation $Y = UbV = Wba$ for the occurrences of Y. Since letter descriptions of type b have a subcontext X the context to be substituted for X cannot have a subcontext of the form b. This is formalized by the instantiation restriction $a < U < V < W < X < b < Y$, which expresses that U, V, W, X may only have occurrences of a whereas Y may have occurrences of a and b. Treating now a and b (resp. U, V, W, X, Y) as constants (resp. word variables), the pair

$$\langle \{X = aaU = VaW, Y = UbV = Wba\}, a < U < V < W < X < b < Y \rangle$$

represents a word multi-equation with regular constraints. One solution is $U = V = W = a$, $X = aaa$, $Y = aba$. Using induction on the linear order $<$ we compute the corresponding solution of T (cf. right-hand side of the above figure; auxiliary bases are ignored). 1. Since a is a ground letter, $\hat{S}(a) = a$; 2. From $X = aaa$ we obtain $\hat{S}(X)$; 3. Now $\hat{S}(b)$ is determined by the form of b and $\hat{S}(X)$; 4. Using $Y = aba$ we obtain $\hat{S}(Y)$. These values completely determine the solution S, which is the solution depicted in Example 2.

The following property of the generalized context problem T was implictly used as a prerequisite for the translation:

P1 It is precisely known which letter descriptions of T are mapped to the same ground context. Letter descriptions that are mapped to the same ground context have exactly the same structure, modulo renaming of superscripts.

This property is needed in the correctness proof. The first part ensures that we know which letter descriptions to translate into the same constant. The role of the second part can be recognized above: during the inductive definition of the context $\hat{S}(L)$ for letter descriptions L of type a (resp. b) no conflict could arise since all these letter descriptions are "isomorphic" and lead to the same result. Yet another, less obvious property is needed as a prerequisite of (Transl4).

P2 Each branching node of two context bases is always labeled.

This condition entails that each unlabeled inner node of T has exactly one child, which is used for the soundness proof. The intermediate translations (Transl2) and (Transl3) are used to transform an arbitrary generalized context problem into one that satisfies conditions P1 and P2. The first, simple step addresses property P2. Call a generalized context problem T *transparent* if it satisfies P2.

Definition 16 (Procedure (Transl2)). *The* input *of this procedure is a generalized context problem T. If there exists a branching point η of two bases of T that is unlabeled, with children $\eta_1, \ldots, \eta_m$, say, then*

1. *non-deterministically choose a function symbol $f \in \Sigma$, of arity $n \geq m$, and label η with f,*

2. *for η introduce n new children in the left-to-right order $\eta'_1, \ldots, \eta'_n$ that are used instead of the old children. Now each old child η_i is nondeterministically either identified with some new child η'_j (which implies that the descendants of η_i are now descendants of η'_j), or it is used as the unique child of some new node η'_j. In this step, distinct new children η'_j are used for distinct old children η_i, and the left-to-right ordering is respected.*
3. *Each new child η'_j that is not used in the previous step represents a leaf. We introduce an individual base* ib of a new type *and define Node(ib) $= \eta'_j$.*
4. *Repeat Steps 1-3 until all branching nodes are labeled. Update the field function accordingly.*

The output *of the procedure consists of the set $\mathcal{T}$ of all generalized context problems that are reached by suitable choices in the nondeterministic steps.*

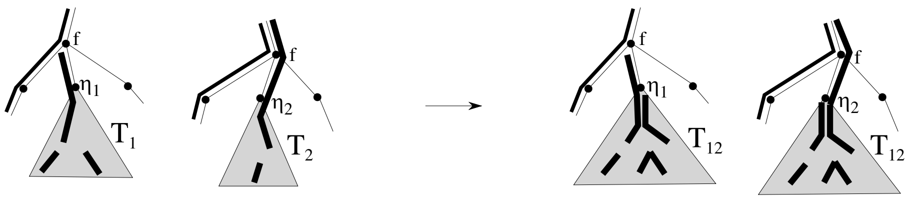
We remark that the "new structural information" for a context base $X^{(i)}$ that is introduced by such a labeling step is not "copied" to other bases $X^{(j)}$. The proof of the following Lemma is straightforward.

Lemma 3 (Transl2). *For each generalized context problem T the algorithm (Transl2) computes a finite set $\mathcal{T}$ of transparent generalized context problems such that T has a solution iff some $T' \in \mathcal{T}$ has a solution. If T has only context bases of k types, then the same holds for the generalized context problems in $\mathcal{T}$.*

Definition 17. *Let $L = (\eta, \eta')$ be a letter description of the generalized context problem T with label f of arity n. Each side node (cf. Def. 7) of (η, η') that is a child of η is called a* top side node *of (η, η'). If $\eta_1, \ldots, \eta_{n-1}$ is the sequence of all top side nodes of L in the natural left-to-right ordering, then η_i is called the* i-th top side node *of L. A node η is called a top side node of T if η is a top side node of a letter description of T. If L and L' are two letter descriptions with the same label and direction, and if η (resp. η') is the i-th top side node of L (resp. L'), then η and η' are called* corresponding *top side nodes of L and L'.*

6 Identification of Letter Descriptions

We now come to the most difficult translation step. In order to reach problems that satisfy Property P1 above, we want to guess which letter descriptions of a given transparent generalized context problem T are mapped to the same ground letter under a given (hypothetical) solution of T, and we want to identify these letter descriptions. We shall proceed in an indirect way and guess (roughly) which top side nodes of T are mapped to identical ground terms. The subtrees of these top side nodes are replaced by a common superposition, which will guarantee that each solution of the new problem will always map these subtrees to the same ground term. This is indicated in the following figure where T_{12} represents the superposition of the subtrees T_1 and T_2 defined by the top side nodes η_1 and η_2:



If, for given letter descriptions L_1 and L_2 of the new problem, each pair of corresponding top side nodes is identified in this sense, this also means that L_1 and L_2 are mapped to the same ground letter under any solution.

Given this idea, there are two major complications. First, a closer look at the technical details shows that we cannot simultaneously treat all equivalence classes of top side nodes that we would like to identify. Basically the difficulties arise from the fact that a letter description may be part of the side area of another letter description. We proceed in an iterative way, identifying top side nodes in a "top-down" manner.

Second, when superimposing the subtrees of distinct top side nodes we may obtain new branching points of context bases in the superposition (in the above figure, two such new branching points are indicated). As a result we obtain (after applying, if necessary, (Transl2) to the superpositioned tree) new letter descriptions and new top side nodes, which are again subject to identification. In general we see no way to guarantee termination. In fact concrete examples with three context variables can be given where (Transl3) in fact does *not* terminate. For input problems with context bases of two types only, the following two observations are used to stop the generation of new branching points.

Observation 1: In a solvable generalized context problem T, branching bases always have distinct type. The proof of this observation is very simple and does not depend on the number of context base types. Intuitively, branching bases of the same type lead to an infinite path in the solution, in other words, to a kind of occur-check inconsistency.

Observation 2: Let T be a transparent generalized context problem with two context base types only. If two letter descriptions L_1 and L_2 of T are identified under a solution S of T, and if η_1 and η_2 are corresponding top side nodes of L_1 and L_2, then the two subproblems T^{η_1} and T^{η_2} can only contain (suffixes of) context bases of one single type. The background, again, is an occur-check problem, the proof is more involved. [2] Since (suffixes of) context bases of one single type cannot branch, a situation as described in the above figure cannot arise.

Before we can give the algorithm, several concepts are needed.

Definition 18. *For $i = 1, 2$, let $T_i = \langle N_i, Lab_i, CB_i, Field_i, IB_i, Node_i\rangle$ be a generalized context problem. T_1 and T_2 are* strictly isomorphic *if there exists a*

[2] The proof—and formulation—of this observation in [17] (cf. Lemma 9.15) has to cope with some additional technical ballast that arises from the introduction of "placeholder" bases and "marker" bases in Steps 1 and 4 of Case III of (Transl3), see below. These bases represent bases of new type.

bijection $F : N_1 \to N_2$ such that F is an embedding (cf. Def. 14) of T_1 in T_2 and F^{-1} is an embedding of T_2 in T_1. Each pair of the form $(\eta, F(\eta))$ ($\eta \in N_1$) is called a pair of corresponding nodes *of T_1 and T_2.*

Definition 19. *Let η be a node of the generalized context problem $T = \langle N, Lab, CB, Field, IB, Node\rangle$. The* subproblem of T defined by η *is the generalized context problem $T^\eta = \langle N^\eta, Lab^\eta, CB^\eta, Field^\eta, IB^\eta, Node^\eta\rangle$ with the following components: N^η is the set of descendants of η, together with η. Lab^η is the restriction of Lab to N^η. Let cb be a context base of CB such that $|Field(cb) \cap N^\eta| \geq 2$. If $Field(cb) \subseteq N^\eta$, then cb is a context base of CB^η with the same field and type as in T. If $Field(cb) \not\subseteq N^\eta$ we introduce a "placeholder" base cb' with field $Field^\eta(cb') := Field(cb) \cap N^\eta$ in CB^η that* represents *the suffix $Field(cb) \cap N^\eta$ of cb. The context base cb' receives a new context variable as its type. For each individual base ib of T with $Node(ib) \in N^\eta$ the set IB^η inherits a base ib of the same type with $Node^\eta(ib) := Node(ib)$.*

Two placeholder bases are depicted in the above figure.

Definition 20. *A transparent generalized context problem T is* marked *if for every atomic subfield φ of the field of a context base of T there exist a context base cb of T such that $Field(cb) = \varphi$.*

The following lemma is the motivation for introducing marker bases in the algorithm.

Lemma 4. *Let η_1 and η_2 be two nodes of the generalized context problem T with solution (t, S). If T is marked and if the problems T^{η_1} and T^{η_2} are strictly isomorphic, then $\hat{S}(\eta_1') = \hat{S}(\eta_2')$ for each pair of corresponding nodes (η_1', η_2') of T^{η_1} and T^{η_2}. In particular $\hat{S}(\eta_1) = \hat{S}(\eta_2)$.*

Definition 21. *Let L and L' be two letter descriptions of the generalized context problem T with the same label and direction. L and L' are* strictly isomorphic *if for all corresponding top side nodes η and η' of L and L' the subproblems T^η and $T^{\eta'}$ are strictly isomorphic.*

Proposition 1. *Let L_1 and L_2 be strictly isomorphic letter descriptions of the marked generalized context problem T. If (t, S) is a solution of T, then $\hat{S}(L_1) = \hat{S}(L_2)$.*

Definition 22. *Let T be a marked generalized context problem. A solution (t, S) of T is* rigid *if it satisfies the following condition for all letter descriptions L_1 and L_2 of T: $\hat{S}(L_1) = \hat{S}(L_2)$ iff L_1 and L_2 are strictly isomorphic.*

Rigid solutions are introduced since in the last translation step only *rigid* solvability of a marked generalized context problem T ensures solvability of its translation.

Definition 23. *Let $T = \langle N, Lab, CB, Field, IB, Node\rangle$ be a transparent generalized context problem. Given an individual variable $x \in X$ we say that $\eta \in N$ is an* x-node of T *if T has an individual base ib of type x with $Node(ib) = \eta$. Let $\mathcal{Y} \subseteq \mathcal{X}$ be a set of individual variables. The subset π of N is called $\mathcal{Y}$-closed if, for each $y \in \mathcal{Y}$ the following condition holds: if π contains an y-node, then π contains each y-node of T.*

In the following procedure we use two sets Π_i and Δ_i. Intuitively, Π_i collects the equivalence classes of nodes that have been identified already, and Δ_i represents the set of all nodes that are still subject to identification. In the sequel, $rel(T)$ denotes the set of all top side nodes and of all nodes of individual bases of T.

Definition 24 (Procedure (Transl3)). *The* input *is a transparent generalized context problem $T = \langle N_0, Lab_0, CB_0, Field_0, IB_0, Node_0\rangle$ with context bases of type X or Y only. Let $\mathcal{X}_0$ denote the set of all individual variables $x \in \mathcal{X}$ such that T has an x-node. In a first step, T is transformed into the problem $T_0 = \langle N_0, Lab_0, CB_0, Field_0, IB_0, Node_0, \Pi_0, \Delta_0\rangle$ where $\Pi_0 := \emptyset$ represents the empty partition and $\Delta_0 := rel(T_0)$.*

I. Assume that the problem $T_i = \langle N_i, Lab_i, CB_i, Field_i, IB_i, Node_i, \Pi_i, \Delta_i\rangle$ is reached after i steps, where $\Pi_i = \{\pi_1, \dots, \pi_i\}$ is a partition of a subset of $rel(T_i)$ and where $\Delta_i = (rel(T_i) \setminus \bigcup \Pi_i)$. If $\Delta_i = \emptyset$, then go to II, otherwise go to III.

II. If T_i is yet not completely marked, then we add appropriate bases of distinct type until a problem T' is reached that is completely marked. Now T' represents an output problem *of (Transl3).*

III. Choose a non-empty subset $\pi_{i+1} = \{\eta_1, \dots, \eta_m\}$ of Δ_i that satisfies the following conditions:

- *π_{i+1} does not contain two labeled nodes with distinct label,*
- *π_{i+1} is a set of maximal elements of Δ_i in the sense that π_{i+1} does not have any element that is a descendant of another node in Δ_i,*
- *π_{i+1} is $\mathcal{X}$-closed.*

If this is not possible, then fail. Otherwise

1. *nondeterministically choose a superposition T^{S_0} of the problems $T_i^{\eta_1}, \dots, T_i^{\eta_m}$ defined by $\eta_1, \dots, \eta_m$.*
2. *If π_{i+1} does not have any x-node for some $x \in \mathcal{X}_0$, then we apply the following* Failure Condition: If T^{S_0} contains two context bases cb_1 and cb_2 such that cb_1 has a node that falls in the side area of cb_2, then fail.
3. *If the superposition T^{S_0} contains any pair of branching bases where the branching node is unlabeled, then introduce a label using the same procedure as in (Transl2). If at this step we introcude new individual bases (cf. Step 3 of (Transl2)), then it is important to use a new type z that is not in $\mathcal{X}_0$. Repeat this until the superposition represents a transparent problem T^{S_1}.*
4. *With each atomic subfield of a context base of T^{S_1} associate a new context base $Z^{(0)}$, using distinct base types Z for distinct fields. Let T^S denote the resulting problem.*

5. *Replace each problem $T_i^{\eta_j}$ $(1 \leq j \leq m)$ by a strictly isomorphic copy T_j^S of T^S. If the (placeholder) base cb' of T^S represents a (suffix of a) base cb in $T_i^{\eta_j}$, then (the suffix of) cb and its corresponding (placeholder) base receive the same field in T_j^S. For each context base of the superposition T^S that is not a placeholder base, including the new bases of the form $Z^{(0)}$ introduced in Step 4, we use a new base of the same type in T_j^S, context bases that represent the same context base of T^S receive corresponding fields in the problems T_j^S. Similarly, for each individual base of T^S we use a new base of the same type in T_j^S, individual bases that represent the same individual base of T^S receive corresponding nodes in the problems T_j^S.*

Let $T_{i+1} = \langle N_{i+1}, Lab_{i+1}, CB_{i+1}, Field_{i+1}, IB_{i+1}, Node_{i+1}, \Pi_{i+1}, \Delta_{i+1}\rangle$ denote the problem obtained in this way where $\Pi_{i+1} := \Pi_i \cup \{\pi_{i+1}\}$ and $\Delta_{i+1} := rel(T_{i+1}) \setminus \bigcup \Pi_{i+1}$. Go to I.

If the input problem T_0 is first-order, then the sets Δ_i only contain $\mathcal{X}_0$-nodes. Then (Transl3) reduces to a first-order unification procedure where the failure condition before Step 1 of Case III corresponds to the clash and occur-check.

Theorem 2. *The procedure (Transl3) terminates. Let $\mathcal{T}$ denote the output set of (Transl3). $\mathcal{T}$ is finite, each $T' \in \mathcal{T}$ is a marked generalized context problem. If in T' two individual bases ib_1 and ib_2, with nodes η_1 and η_2, say, have the same type, then the subproblems T'^{η_1} and T'^{η_2} are strictly isomorphic. Moreover, if T has a solution, then there exists a problem $T' \in \mathcal{T}$ such that T' has a rigid solution. Conversely, if some $T' \in \mathcal{T}$ has a solution, then T is solvable.*

We note that termination is just a consequence of the Failure Condition of Step 2 of Case III of (Transl3). This condition, justifiable using the above Observations 1 and 2, cannot be used if T has context bases of three distinct types.

7 Summing Up

We are now able to prove the Main Theorem (Theorem 1). Here we only treat the case where we just have one input equation. Let $s = t$ be a context equation with two context variables. Combining the results of Lemma 2, Lemma 3, Theorem 2, and a corresponding Lemma for (Transl4) that is omitted here it follows that we may effectively compute a finite set M of systems of word equations with regular constraints such that $s = t$ has a solution if and only if a system in M has a solution. The results in [18] show that solvability of M is decidable. □

References

1. H. Comon. Completion of rewrite systems with membership constraints, part I: Deduction rules and part II: Constraint solving. Technical Report, CNRS and LRI, Université de Paris Sud, 1993. to appear in JSC.
2. W. Farmer. Simple second-order languages for which unification is undecidable. *Theoretical Computer Science*, 87:173–214, 1991.

3. H. Ganzinger, F. Jacquemard, and M. Veanes. Rigid reachability. In J. Hsiang and A. Ohori, editors, *Advances in Computing Science - ASIAN'98*, Springer LNCS 1538, pages 4–21, 1998.
4. W. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
5. A. Kościelski and L. Pacholski. Complexity of Makanin's algorithms. *Journal of the Association for Computing Machinery*, 43:670–684, 1996.
6. J. Levy. Linear second order unification. In *Proc. of the 7th Int. Conf. on Rewriting Techniques and Applications*, Springer LNCS 1103, pages 332–346, 1996.
7. J. Levy and M. Veanes. On the undecidability of second-order unification. *Submitted to Information and Computation*, 1999.
8. G. Makanin. The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik*, 32(2):129–198, 1977.
9. J. Marcinkowski. Undecidability of the first order theory of one-step right ground rewriting. In H. Comon, editor, *International Conference on Rewriting Techniques and Applications*, Springer LNCS 1232, pages 241–253, 1997.
10. J. Niehren, M. Pinkal, and P. Ruhrberg. On equality up-to constraints over finite trees, context unification, and one-step rewriting. In *Proc. of the Int. Conf. on Automated Deduction*, Springer LNCS 1249, pages 34–48, 1997.
11. J. Niehren, M. Pinkal, and P. Ruhrberg. A uniform approach to underspecification and parallelism. Technical Report, 1997.
12. J. Niehren, S. Tison, and R. Treinen. On stratified context unification and rewriting constraints. Talk at CCL'98 Workshop, 1998.
13. M. Schmidt-Schauß. Unification of stratified second-order terms. Internal Report 12/94, Fachb. Informatik, J.W. Goethe-Universität Frankfurt, Germany, 1994.
14. M. Schmidt-Schauß. An algorithm for distributive unification. *Theoretical Computer Science*, 208:111–148, 1998.
15. M. Schmidt-Schauß. Decidability of bounded second order unification. Draft, Fachbereich Informatik, J.W. Goethe-Universität Frankfurt, Germany, 1998.
16. M. Schmidt-Schauß and K. U. Schulz. On the exponent of periodicity of minimal solutions of context equations. In *Rewriting Techniques and Applications, Proc. RTA'98*, volume 1379 of *LNCS*, pages 61–75. Springer-Verlag, 1998.
17. M. Schmidt-Schauß and K. U. Schulz. Solvability of context equations with two context variables is decidable. CIS-Report 98-114, CIS, University of Munich, Germany, 1999. available under `ftp://ftp.cis.uni-muenchen.de/pub/cis-berichte/CIS-Bericht-98-114.ps`.
18. K. U. Schulz. Makanin's algorithm - two improvements and a generalization. In *Proc. of IWWERT 1990*, Springer LNCS 572, pages 85–150, 1990.
19. K. U. Schulz. Word unification and transformation of generalized equations. *J. Automated Reasoning*, pages 149–184, 1993.
20. R. Treinen. The first-order theory of one-step rewriting is undecidable. In H. Ganzinger, editor, *7th International Conference on Rewriting Techniques and Applications*, Springer LNCS 1103, pages 276–286, Rutgers University, NJ, USA, 1996.
21. S. Vorobyov. The first-order theory of one step rewriting in linear noetherian systems is undecidable. In H. Comon, editor, *International Conference on Rewriting Techniques and Applications*, Springer LNCS 1232, pages 254–268, 1997.
22. S. Vorobyov. The $\forall\exists^8$-equational theory of context unification is co-recursively enumerable hard. Talk at CCL'98 Workshop, 1998.

Complexity of the Higher Order Matching

ToMasz Wierzbicki[⋆]

University of Wrocław

Abstract. We use the standard encoding of Boolean values in simply typed lambda calculus to develop a method of translating SAT problems for various logics into higher order matching. We obtain this way already known NP-hardness bounds for the order two and three and a new result that the fourth order matching is NEXPTIME-hard.

1 Introduction

Consider two normalized simply typed lambda terms M and N, where N is closed (does not contain free variables). The higher order matching problem $M \stackrel{?}{=} N$ (also known as pattern matching,[1] range problem or instantiation problem) is to decide whether there exists a substitution θ for free variables in M, such that $M\theta$ is $\beta\eta$-reducible to N. Matching is a special case of *unification*, where the restriction that N is closed is removed (and a solution of $M \stackrel{?}{=} N$ is a substitution θ such that $M\theta$ and $N\theta$ are equal modulo $\beta\eta$-conversion). The order of a problem $M \stackrel{?}{=} N$ is the highest functionality order of free variables occurring in M. At the time of writing this paper the decidability of higher order matching with unbounded order is still an open question (for 23 years since the problem has been explicitly stated for the first time and conjectured decidable [11]) and seems to be very difficult. This is why it is of interest to investigate the problem with the order bounded by a constant.

Higher order matching was first studied by Lewis D. Baxter and Gerard Huet in the mid-seventies in their PhD theses [2,11]. Baxter [2] showed that second order matching is NP-complete by reduction to the "one-in-three" SAT problem. At the same time this case was independently investigated by Huet [11,12]. In 1975 Huet devised a semi-decision algorithm for (undecidable even in the second order case) unification problem [10]. Later an analysis of behavior (termination, running time, *etc.*) of this algorithm and its modifications was a source of a couple of results about decidability and complexity of some special cases of matching.

In 1992 Gilles Dowek [6] proved decidability of third order matching and suggested how to control Huet's algorithm to obtain a doubly exponential decision procedure. At the same time David Wolfram [25,26] gave a proof of NP-hardness of third order pure (*i.e.*, without constants) matching. Surprisingly it turned out

⋆ Supported by KBN grant 8 T 11C 029 13.

1 Since the term "pattern" has multiple meanings in the theory of higher order unification, *e.g.*, the one introduced by Dale Miller [15], we would rather avoid it when speaking about unification of an arbitrary term with a closed (or ground) one.

order	decidable?	lower bound	upper bound
1	yes [18]	?	NLOGSPACE [8]
2	yes [12]	NP-complete [2,12,5]	
3	yes [6]	NP-complete [25,5]	
4	yes [16]	NEXPTIME-hard	2-NEXPTIME (?) [5]
ω	?	not elementary recursive [24]	?

Table 1. Decidability and complexity of higher order matching

that it was an exact bound: Hubert Comon and Yan Jurski [5] designed an NP algorithm for that problem. In the same paper they again showed a proof of NP-hardness of second order case with at least one binary constant by reduction to the 3-SAT problem. They also presented an algorithm for the fourth order case (the case shown to be decidable a year before by Padovani [16]), and suggested how it may probably be improved to be run in 2-NEXPTIME (however it was not formally proved). We show this case is NEXPTIME-hard.

In the seventies higher order matching (under a name "the range problem") was also investigated by Richard Statman [22], who reduced its decidability to decidability of lambda definability. Moreover, the decidability of higher order matching with so called *delta functions* is equivalent to the decidability of lambda definability. Unfortunately Ralf Loader [13] gave a negative answer to the later question. Thus matching with delta functions is undecidable. Some other extensions of matching are also known to be undecidable, *e.g.*, matching in Gödel's system T and Girard's system F [7]. Statman is also the author of a theorem stating that deciding equality of simply typed lambda terms is not elementary recursive [21,14]. This result can be immediately used to prove the same lower bound for higher order matching, as noted by Sergei Vorobyov [24].

All the mentioned facts are summarized in Table 1.

Statman [23] showed that any k-th order unification problem containing constants of order not greater than m may be transformed to one of order $\max(k, m+1)$ under an empty signature. Thus in general constants do not alter the complexity of matching. However for the second order case the lower bound is in [5] proved under the assumption that there is at least one binary second order constant. Removing this constant using Statman's construction results in a pure problem, but of the third order. The proof may be however in an obvious way changed to apply also to the pure language. Thus all lower bounds shown in Table 1 apply to pure languages (without constants), while upper bounds hold for arbitrary signatures.

In this paper we show that although the second and third order matching are both NP-complete, the third order case is somewhat harder in the following sense: the second order matching is in PTIME if the number of unknowns is bounded by a constant, while the third order case is NP-hard even with only one unknown variable.

The paper is self-contained. In the next section we recall all needed definitions and basic facts about simply typed lambda calculus and higher order unification. Section 3 contains a detailed description of our method of coding SAT problems for various logics in higher order matching. Section 4 contains a proof of the main result of the paper, *i.e.*, a proof of NEXPTIME-hardness of the fourth order matching.

2 Definitions and Basic Facts

Types are built over one *base type* o, that is o is a type, and if σ and τ are types, then $(\sigma \to \tau)$ also is. Small Greek letters $\sigma, \tau, \ldots$ stand for arbitrary types. To drop parentheses we assume that $\to$ associates to the right, *i.e.*, $\sigma \to \tau \to \rho$ means $\sigma \to (\tau \to \rho)$. The order of a type is defined as follows:

$$\begin{aligned}\mathrm{order}(o) &= 1\\ \mathrm{order}(\sigma \to \tau) &= \max(\mathrm{order}(\sigma)+1, \mathrm{order}(\tau))\end{aligned}$$

For every type σ there is an infinite set $\mathbb{V}_\sigma$ of *term variables of type* σ. Arbitrary elements of $\mathbb{V}_\sigma$ are denoted by $x^\sigma, y^\sigma, \ldots$ We assume that $\mathbb{V}_\sigma$ are disjoint for different types σ. Besides variables one sometimes considers pairwise disjoint, and disjoint with the sets of variables, possibly empty sets $\mathbb{C}_\sigma$ of *constants of type* σ. The union of $\mathbb{C}_\sigma$ for all types σ is called *a signature.* The formation rules for *lambda terms* (or *terms* for short) are the following: a variable $x^\sigma \in \mathbb{V}_\sigma$ is a term of type σ. Similarly a constant $c \in \mathbb{C}_\sigma$ is a term of type σ. If M is a term of type $\sigma \to \tau$ and N is a term of type σ, then (MN) is a term of type τ. If $x^\sigma \in \mathbb{V}_\sigma$ and M is a term of type τ, then $(\lambda x^\sigma.M)$ is a term of type $\sigma \to \tau$. We use the letters $M, N, \ldots$ to denote arbitrary lambda terms. For better readability and to save space we drop a type superscript from every bound occurrence of a term variable, and parentheses using the following rule: $\lambda x_1^{\sigma_1} \ldots \lambda x_n^{\sigma_n}.M_1 M_2 \ldots M_m$ (or even shorter $\lambda \vec{x}.M_1 M_2 \ldots M_m$) stands for $(\lambda x_1^{\sigma_1}.(\ldots(\lambda x_n^{\sigma_n}.(\ldots(M_1 M_2) \ldots M_m))))$. Notation $M^k N$ stands for the term

$$\underbrace{M(M(\ldots(M}_{k\ \text{times}} N)\ldots))$$

and $\sigma^k \to \tau$ for the type

$$\underbrace{\sigma \to \cdots \to \sigma}_{k\ \text{times}} \to \tau$$

We sometimes write $\sigma_1, \ldots, \sigma_n \to \tau$ instead of $\sigma_1 \to \cdots \to \sigma_n \to \tau$ even if τ is not a base type o, and $M(N_1, \ldots, N_n)$ instead of $MN_1 \ldots N_n$. This notation is usually connected with so called η-long form. We however do *not* assume that lambda terms are in η-long form (we even do not define such a notion).

The notions of free (denoted $\mathrm{FV}(\cdot)$) and bound variables of a term, type preserving (simultaneous) substitution (denoted $[x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}/M_1, \ldots, M_n]$ or shortly $[x_i^{\sigma_i}/M_i]_{i=1}^n$), α-conversion, β- and η-reduction, $\beta\eta$-equality (denoted $=_{\beta\eta}$), and the normal form of a term (denoted $\mathrm{NF}(\cdot)$) are defined as usual (see, *e.g.*, [1]). A term is *closed* if it does not contain free variables.

The notion of order is extended to lambda terms: the order of a term is the highest order of types of its subterms. In particular the order of a variable x^σ is the order of its type σ.

Note that lambda terms may have types that are of exponential size comparing to the size of type-erased terms, and in our formulation term variables are explicitly decorated with their types. Thus special care is needed in order to avoid an exponential blow-up of the term size. An elegant way of doing this is by representing types as directed acyclic graphs rather than trees, since the dag-size of types is always polynomial in the size of type-erased lambda terms. However if the order of types is bounded by a constant, then even the tree size of types is polynomial.

A *unification problem* is a pair of terms of the same type, in normal form, possibly containing free variables, written $M \stackrel{?}{=} N$. A *solution* to a problem $M \stackrel{?}{=} N$ is a substitution θ, such that $M\theta =_{\beta\eta} N\theta$. The order of a problem $M \stackrel{?}{=} N$ is the highest order of free variables occurring in M and N. Instead of speaking about one equation $M \stackrel{?}{=} N$, one can equivalently speak about a collection of equations $\{M_i \stackrel{?}{=} N_i\}_{i=1}^k$. The later one is solvable exactly when a single equation

$$\lambda y.y(M_1,\ldots,M_k) \stackrel{?}{=} \lambda y.y(N_1,\ldots,N_k)$$

is. A unification problem under an empty signature is called *a pure problem.* A *matching problem* is a unification problem $M \stackrel{?}{=} N$ in which a term N is closed. Even second order unification in the presence of at least one binary symbol is undecidable [9]. To the contrary, matching is known to be decidable up to the order of four and conjectured decidable with unrestricted order [11].

The *cardinality* of a type σ, notation: $\mathrm{card}(\sigma)$, is the number of its normal inhabitants, *i.e.*, the number of closed terms in normal form of that type. A type σ is *inhabited* if there exists a closed term of that type, *i.e.*, if $\mathrm{card}(\sigma) > 0$.

Fact 1 (Statman [21,20]). *Consider lambda terms under an empty signature (*i.e.*, without constants). Given a type σ.*

1. *If* $\mathrm{order}(\sigma) = 1$, *then* $\mathrm{card}(\sigma) = 0$.
2. *If* $\mathrm{order}(\sigma) = 2$, *then* $\sigma = o^k \to o$ *for some* $k \geq 1$, *and* $\mathrm{card}(\sigma) = k$.
3. *If* $\mathrm{order}(\sigma) \geq 3$, *then* $\mathrm{card}(\sigma) = 0$ *or* $\mathrm{card}(\sigma) = \infty$.
4. *Moreover, a type $\sigma = \tau \to \rho$ is inhabited, if and only if τ is not inhabited or ρ is inhabited.*

Thus cardinality of a type may be calculated in DLINTIME.[2]

Indeed, if $\mathrm{order}(\sigma) = 1$, then $\sigma = o$. In the absence of constants all closed terms are functions, *i.e.*, are of order greater than one, thus $\mathrm{card}(\sigma) = 0$.

[2] As opposed to the Curry style (implicitly typed) lambda calculus or a calculus with an infinite number of base types, for which this problem is PSPACE-complete.

To see part 2, note that there are exactly k different (modulo β- or $\beta\eta$-conversion) closed terms of type $o^k \to o$, and they are of the form $\lambda u_1^o \dots \lambda u_k^o.u_j$, for $j = 1, \dots, k$ (providing there are no constants in the language). Indeed, consider the β-normal form of a term of type $o^k \to o$. It starts with a binder $\lambda u_1^o \dots \lambda u_i^o$ and then contains a body of type $o^{k-i} \to o$, which is not an abstraction. Consider the head of the body. It has to be one of the variables from the binder, since there are no constants and the term is closed. Since every variable u_j is of base type o, *i.e.*, is not a function, it cannot be applied to anything, and has to constitute the whole body of the term. Thus the term is of the form $\lambda u_1^o \dots \lambda u_i^o.u_j$. Finally, since its type is $o^k \to o$, it should be $i = k$.

To prove part 3, it is sufficient to show that every inhabited type σ of order$(\sigma) \geq 3$ possesses *a splinter* (see [21]), that is a pair of terms: M of type $\sigma \to \sigma$ and N of type σ, such that all terms $M^n N$ for $n \geq 0$ are not $\beta\eta$-equal. Indeed, as N choose any term in normal form of type σ. There exists at least one, since σ has been assumed to be inhabited. The type σ may be uniquely written in the form $\sigma_1 \to \dots \to \sigma_i \to o$. Since order$(\sigma) \geq 3$, there exists $j \leq i$, such that $\sigma_j \neq o$, that is σ_j is a function type. Let $\sigma_j = \tau_1 \to \dots \to \tau_k \to o$. A term $x^\sigma(y_1^{\sigma_1}, \dots, y_i^{\sigma_i})$ is of base type o. By prefixing it with an appropriate lambda binder $\lambda\vec{z}.$ we may build terms of any type (although not closed, since containing free occurrences of variables x^σ and $y_1^{\sigma_1}, \dots, y_i^{\sigma_i}$), in particular of types $\tau_1, \dots, \tau_k$. A term

$$M = \lambda x^\sigma.\lambda y_1^{\sigma_1} \dots \lambda y_i^{\sigma_i}.y_j\big(\lambda\vec{z}_1.x(y_1, \dots, y_i), \dots, \lambda\vec{z}_k.x(y_1, \dots, y_i)\big)$$

together with N forms a splinter. For example for type $\mathfrak{N} = (o \to o) \to o \to o$ the term M is $\lambda n^{\mathfrak{N}}.\lambda f^{o\to o}.\lambda x^o.f(n(f, x))$. Choosing for N any term of type $\mathfrak{N}$, say $\lambda f^{o\to o}.\lambda x^o.x$ we may generate inifitely many terms $M^k N =_{\beta\eta} \lambda f^{o\to o}.\lambda x^o.f^k x$, where $k \geq 0$, of that type (incidentally they happen to be all the inhabitants of $\mathfrak{N}$). Note this way we are able to list an infinite number of normal inhabitants of a type, if we have one. With a little more care we might list this way, as in the last example, *all* inhabitants of a type. Huet's semidecision procedure for higher order unification [10] is in fact based on such a systematic way of listing all normal inhabitants of types of unknown variables and checking whether some of them are solutions to a given unification problem.

The part 4 of Fact 1 may easily be proved by induction on the structure of a given type.

Fact 1 implies decidability of pure second order unification. Indeed, we have a naïve NP algorithm for a second order unification problem $M \stackrel{?}{=} N$: guess a solution θ and check whether $M\theta =_{\beta\eta} N\theta$ by reducing both sides to their normal forms, which may be computed in polynomial time. To see this let M be a term in β-normal form, perhaps containing a free variable $x^{o^k \to o}$, and N be a term in β-normal form of type $o^k \to o$. Then the normal form of $M[x^{o^k \to o}/N]$ can be computed in polynomial time. Indeed, M may contain subterms of the form $x^{o^k \to o}(M_1, \dots, M_i)$, where $i \leq k$ and terms $M_1, \dots, M_i$ are of type o. Substituting N for $x^{o^k \to o}$ leads to a redex $N(M_1, \dots, M_i)$ (providing that N starts with an abstraction), but contracting it cannot create new redexes, since the arguments $M_1, \ldots, M_i$ being of type o are not functions and cannot be applied to anything.

Note however that if the number of variables occurring in a problem is bounded by a constant, the problem is soluble in PTIME: list all possible solutions and check them deterministically. Thus:

Corollary 1. *Second order pure unification (in particular matching) is in NP. Second order pure unification (in particular matching) with bounded number of variables is in PTIME.*

In the next section we show corresponding lower bounds for these problems.

Although second order pure matching problems have always only a finite number of different solutions (since there is only a finite number of different terms of a second order type), their solutions may not be unique, *e.g.*, all terms of type $o^k \to o$ except $\lambda v_1^o \ldots \lambda v_k^o.v_1$ are solutions to

$$\lambda u_1^o.\lambda u_2^o.x^{o^k \to o}(u_1, u_2, \ldots, u_2) \stackrel{?}{=} \lambda u_1^o.\lambda u_2^o.u_2$$

Since the substitutions $[x^{o^k \to o}/\lambda v_1^o \ldots \lambda v_k^o.v_i]$ for $i = 2, \ldots, k$ are unrelated, there is no here, unlike for the first order unification, the most general unifiers (however a notion of *a complete set of* the most general unifiers may be defined; in higher order cases these sets may be infinite).

Fact 1 suggests that third and higher order matching problems may have even an infinite number of unrelated solutions, since any type of order greater than two, if inhabited at all, possesses infinitely many normal inhabitants. For example every substitution

$$[f^{(o \to o) \to o \to o}/\lambda z^{o \to o}.\lambda y^o.z^k y]$$

for $k \geq 0$, is a solution to

$$\lambda y^o.f^{(o \to o) \to o \to o}(\lambda x^o.x, y) \stackrel{?}{=} \lambda y^o.y$$

Both sides of an instance of a matching problem $M \stackrel{?}{=} N$ should be in normal form, hence it may suggest, that computational complexity of the β-reduction process cannot show off in solving that problem. However redexes may stay latent until searching for solutions begins. Thus it turns out that matching is at least as hard as testing equality of simply typed lambda terms, which is known to be non-elementary:

Theorem 1 (Statman [21,14]). *Given two simply typed closed lambda terms M and N, the second one in normal form. It is not elementary recursive to decide whether N is a normal form of M.*

This theorem immediately implies:

Corollary 2 (Vorobyov [24]). *Higher order pure matching is not elementary recursive.*

Since terms M and N in a matching problem $M \overset{?}{=} N$ should be in normal form, all we need is just to "hide" all redexes. Given two terms M and N, the second one in normal form. Let $s^{(\sigma\to\tau),\sigma\to\tau}$ be fresh variables (s like "suspend", since it resembles the suspension operator used in eager programming languages to simulate lazy evaluation), where σ and τ range over all such pairs of types, that there exists a subterm $(M_1 M_2)$ of M, such that $M_1 : \sigma \to \tau$ and $M_2 : \sigma$. Let

$$\mathcal{S} = \left\{ s^{(\sigma\to\tau),\sigma\to\tau} \overset{?}{=} \lambda x^{\sigma\to\tau}.\lambda y^{\sigma}.x(y) \right\}$$

Transform M into M^* "hiding" all β-redexes as follows (η-redexes may be "hidden" in a similar way):

$$\begin{aligned} x^* &= x \\ (\lambda x.M)^* &= \lambda x.M^* \\ (M_1 M_2)^* &= s^{(\sigma\to\tau),\sigma\to\tau}(M_1^*, M_2^*), \quad \text{where } M_1 : \sigma \to \tau \text{ and } M_2 : \sigma \end{aligned}$$

The set of equations $\mathcal{S}$ as well as the term M^* may be generated in time polynomial on the size of M. Clearly $\mathcal{S} \cup \{M^* \overset{?}{=} N\}$ is a well-formed instance of higher order matching (of unbounded order, since the order of $s^{(\sigma\to\tau),\sigma\to\tau}$ may be arbitrarily large), which has a solution exactly when N is a normal form of M. Thus:

Fact 2. *$(k+1)$-st order matching is as hard as testing equality of simply typed lambda terms of order at most k.*

We use roughly the same technique in coding SAT problems in the next section: we write lambda terms for which testing equality is hard and "hide" redexes in the way described above.

It is hard to find out if all computational power inherent in matching is essentially the β-reduction, or there is something more in it. It is easy to see that we are able to achieve nondeterministic lower bounds by leaving some objects to be "guessed" and substituted for free variables. From the other hand however this trick does not seem to help much more than simply reduce the order of unknown variables.

3 Coding SAT Problems in Higher Order Matching

In this section we develop a method of coding SAT problems for various logics in higher order matching.

Propositional formulas are expressions built over an infinite set of propositional variables $\alpha, \beta, \ldots$ using unary negation symbol $\neg$ and binary connectives $\vee$ and $\wedge$. When the quantification $(\forall, \exists)$ over propositional variables is allowed, the expressions are called *quantified Boolean formulas.* The satisfiability problem (SAT for short) for propositional formulas is NP-complete, and for quantified Boolean formulas — PSPACE-complete [17].

Usually the Boolean values and logical connectives are encoded in the lambda calculus in the following way:

$$\begin{aligned}\mathfrak{B} &= o, o \to o\\ \mathit{true} &= \lambda u^o.\lambda v^o.u\\ \mathit{false} &= \lambda u^o.\lambda v^o.v\\ \mathit{and}[M, N] &= \lambda u^o.\lambda v^o.M(N(u, v), v)\\ \mathit{or}[M, N] &= \lambda u^o.\lambda v^o.M(u, N(u, v))\\ \mathit{not}[M] &= \lambda u^o.\lambda v^o.M(v, u)\end{aligned}$$

We use the abbreviations defined above to build an instance of the third order matching problem:

$$\begin{aligned}\mathcal{B} \;=\; \Big\{ & a^{\mathfrak{B},\mathfrak{B}\to\mathfrak{B}} \overset{?}{=} \lambda x^{\mathfrak{B}}.\lambda y^{\mathfrak{B}}.\mathit{and}[x, y],\\ & r^{\mathfrak{B},\mathfrak{B}\to\mathfrak{B}} \overset{?}{=} \lambda x^{\mathfrak{B}}.\lambda y^{\mathfrak{B}}.\mathit{or}[x, y],\\ & n^{\mathfrak{B}\to\mathfrak{B}} \overset{?}{=} \lambda x^{\mathfrak{B}}.\mathit{not}[x]\Big\}\end{aligned}$$

The only solution of $\mathcal{B}$ substitutes the corresponding right hand sides for the free variables $a^{\mathfrak{B},\mathfrak{B}\to\mathfrak{B}}$, $r^{\mathfrak{B},\mathfrak{B}\to\mathfrak{B}}$ and $n^{\mathfrak{B}\to\mathfrak{B}}$ (by definition a solution to $x^\sigma \overset{?}{=} M$, where M is closed, should substitute for x^σ a term M', such that $M' =_{\beta\eta} M$; since both M and M' should be in normal form, which is unique, $M = M'$, and the only solution to $x^\sigma \overset{?}{=} M$ is $[x^\sigma/M]$).

We will use these variables to define other parts of the constructed matching problem. Note that now we are ready to reproof the following (well known [2,25,5]):

Fact 3. *Third order pure matching is NP-hard.*

Indeed, code the SAT problem for propositional formulas, *i.e.*, transform any formula in the following way: for every propositional variable α choose a fresh term variable $x_\alpha^{\mathfrak{B}}$. Define a transformation $\cdot^*$ as follows:

$$\begin{aligned}\alpha^* &= x_\alpha^{\mathfrak{B}}\\ (\phi \wedge \psi)^* &= a^{\mathfrak{B},\mathfrak{B}\to\mathfrak{B}}(\phi^*, \psi^*)\\ (\phi \vee \psi)^* &= r^{\mathfrak{B},\mathfrak{B}\to\mathfrak{B}}(\phi^*, \psi^*)\\ (\neg\phi)^* &= n^{\mathfrak{B}\to\mathfrak{B}}(\phi^*)\end{aligned}$$

Clearly for any propositional formula the third order matching problem

$$\mathcal{B} \cup \{\phi^* \overset{?}{=} \mathit{true}\}$$

can be constructed in time polynomial in the size of ϕ, and has a solution exactly when ϕ is satisfiable.

Note that variables that play the essential rôle in the coding presented above are the second order variables of type $\mathfrak{B}$, which encode Boolean values. Third order variables are auxiliary. Indeed, take a variable $x^{\mathfrak{B}}$. There are only two closed terms in normal form of type $\mathfrak{B}$, *i.e.*, *true* and *false*. The only solution of

$$\lambda u^o.\lambda v^o.x^{\mathfrak{B}}(u,v) \overset{?}{=} \lambda u^o.\lambda v^o.u$$

is *true*. Similarly, the solution of

$$\lambda u^o.\lambda v^o.x^{\mathfrak{B}}(v,u) \overset{?}{=} \lambda u^o.\lambda v^o.u$$

is *false*. If we have two variables $x^{\mathfrak{B}}$ and $y^{\mathfrak{B}}$, then the equation

$$\lambda u^o.\lambda v^o.x^{\mathfrak{B}}(y^{\mathfrak{B}}(v,u), y^{\mathfrak{B}}(u,v)) \overset{?}{=} \lambda u^o.\lambda v^o.u$$

has two solutions substituting for $x^{\mathfrak{B}}$ and $y^{\mathfrak{B}}$ two different values of type $\mathfrak{B}$, *i.e.*, the last equation forces $x^{\mathfrak{B}}$ to be the negation of $y^{\mathfrak{B}}$ and *vice versa*. In a similar way we are able to encode other Boolean functions. Now take a propositional formula ϕ. For every propositional variable α choose a fresh term variable $x^{\mathfrak{B}}_\alpha$, and for every compound (not being a single variable) subformula ψ of ϕ choose a fresh term variable[3] $x^{\mathfrak{B}}_\psi$, and define lambda terms M_ψ in the following way:

$$\begin{aligned}
M_{\psi\wedge\rho} &= \lambda u^o.\lambda v^o.x^{\mathfrak{B}}_{\psi\wedge\rho}(x^{\mathfrak{B}}_\psi(x^{\mathfrak{B}}_\rho(u,v),v), x^{\mathfrak{B}}_\psi(x^{\mathfrak{B}}_\rho(v,u),u))\\
M_{\psi\vee\rho} &= \lambda u^o.\lambda v^o.x^{\mathfrak{B}}_{\psi\vee\rho}(x^{\mathfrak{B}}_\psi(u,x^{\mathfrak{B}}_\rho(u,v)), x^{\mathfrak{B}}_\psi(v,x^{\mathfrak{B}}_\rho(v,u)))\\
M_{\neg\psi} &= \lambda u^o.\lambda v^o.x^{\mathfrak{B}}_{\neg\psi}(x^{\mathfrak{B}}_\psi(v,u), x^{\mathfrak{B}}_\psi(u,v))
\end{aligned}$$

Observe that a substitution θ is a solution of $M_{\phi\wedge\psi} \overset{?}{=} \mathit{true}$ if and only if $x_\phi\theta =_{\beta\eta} \mathit{true}$ and $x_\psi\theta =_{\beta\eta} \mathit{true}$. Similarly for other Boolean connectives. Thus the second order matching problem:

$$\left\{M_\psi \overset{?}{=} \mathit{true} \mid \psi \text{ subformula of } \phi, \text{ not a variable}\right\} \cup \left\{\lambda u^o.\lambda v^o.x^{\mathfrak{B}}_\phi(u,v) \overset{?}{=} \mathit{true}\right\}$$

is soluble exactly when the formula ϕ is satisfiable. The last equation restricts the set of substitutions to those making the whole formula ϕ true. Thus we have (implicit in [5]):

Fact 4. *Second order pure matching is NP-hard.*

Recall that second order pure matching with bounded number of variables is in PTIME. To the contrary, third order case is NP-hard even with only one variable. Indeed, let $\{x^{\mathfrak{B}}_1, \ldots, x^{\mathfrak{B}}_n\}$ be the set of free variables occurring in the previous coding. Choose one fresh third order variable $x^{(o^n\to o)\to\mathfrak{B}}$, and substitute

$$x^{(o^n\to o)\to\mathfrak{B}}(\lambda u^o_1 \ldots \lambda u^o_n.u_i)$$

[3] It is irrelevant whether two identical subformulas are assigned the same term variable or not. It may depend on the way a formula is represented: as a tree or as a directed acyclic graph. It is however essential to ensure that all occurrences of the same propositional variable α are associated with the same term variable $x^{\mathfrak{B}}_\alpha$.

for the variable $x_i^{\mathfrak{B}}$ in the previous coding. The resulting third order problem contains only one variable $x^{(o^n \to o) \to \mathfrak{B}}$ and is soluble exactly when the previous one is. To see this, let θ be a solution to the previous system. Then

$$[x^{(o^n \to o) \to \mathfrak{B}} \,/\, \mathrm{NF}(\lambda s^{o^n \to o}.\lambda u^o.\lambda v^o.s((x_1^{\mathfrak{B}}\theta)uv, \ldots, (x_n^{\mathfrak{B}}\theta)uv))]$$

satisfies the transformed one, where $\mathrm{NF}(N)$ is the normal form of N. Conversely if $[x^{(o^n \to o) \to \mathfrak{B}}/M]$ is a substitution satisfying the transformed problem, then

$$[x_i^{\mathfrak{B}} \,/\, \mathrm{NF}(M(\lambda u_1^o \ldots \lambda u_n^o.u_i))]_{i=1}^n$$

satisfies the original system. This is because the β-reduction is Church-Rosser. Thus we have:

Fact 5. *Third order pure matching with only one variable is NP-hard.*

It might be interesting to note, that the same trick applies to the undecidability proof of the third order unification problem. Statman [23] recalls that third order pure unification with nine variables of type $\mathfrak{N} = (o \to o) \to o \to o$ is undecidable (by encoding of the 10th Hilbert problem with nine variables), and that it is not known whether the problem with eight variables is decidable or not. Although we do not know the answer to the last question, it is easy to see that third order unification problem with only one variable of type $(o^9 \to o) \to \mathfrak{N}$ is undecidable.

Now let us get back to the first coding $\cdot^*$ described in this section. It can easily be extended to quantified Boolean formulas at the price of raising the order by one. Indeed, a formula $\forall\alpha.\phi(\alpha)$ means $\phi(\mathit{true}) \wedge \phi(\mathit{false})$, and $\exists\alpha.\phi(\alpha)$ means $\phi(\mathit{true}) \vee \phi(\mathit{false})$, thus define

$$\begin{aligned} \mathcal{Q}_{\mathfrak{B}} \;=\; \Big\{ & f^{(\mathfrak{B}\to\mathfrak{B})\to\mathfrak{B}} \overset{?}{=} \lambda g^{\mathfrak{B}\to\mathfrak{B}}.\mathit{and}[g(\mathit{true}), g(\mathit{false})], \\ & e^{(\mathfrak{B}\to\mathfrak{B})\to\mathfrak{B}} \overset{?}{=} \lambda g^{\mathfrak{B}\to\mathfrak{B}}.\mathit{or}[g(\mathit{true}), g(\mathit{false})] \Big\} \end{aligned}$$

The variables $f^{(\mathfrak{B}\to\mathfrak{B})\to\mathfrak{B}}$ and $e^{(\mathfrak{B}\to\mathfrak{B})\to\mathfrak{B}}$ are of the fourth order. Now extend the translation $\cdot^*$ as follows:

$$\begin{aligned} (\forall\alpha.\phi)^* &= f^{(\mathfrak{B}\to\mathfrak{B})\to\mathfrak{B}}(\lambda x_\alpha^{\mathfrak{B}}.\phi^*) \\ (\exists\alpha.\phi)^* &= e^{(\mathfrak{B}\to\mathfrak{B})\to\mathfrak{B}}(\lambda x_\alpha^{\mathfrak{B}}.\phi^*) \end{aligned}$$

It is evident that the fourth order matching problem $\mathcal{B} \cup \mathcal{Q}_{\mathfrak{B}} \cup \{\phi^* \overset{?}{=} \mathit{true}\}$ has a solution exactly when a quantified Boolean formula ϕ is satisfiable. Finally we have another (still not very exciting):

Fact 6. *Fourth order pure matching is PSPACE-hard.*

Note that this construction does not actually make use of the fact that there may be some unknowns left to be guessed as a part of the solution in a matching problem. Complexity of the problems constructed above is caused just by complexity of finding the normal form of a lambda term. To see this take a quantified

Boolean formula ϕ. If it is not closed, write an appropriate number of existential quantifiers at its beginning to bind all free propositional variables occurring in it. Use the translation $\cdot^*$ from the previous construction. The lambda term ϕ^* is of the third order and contains two free variables: $f^{(\mathfrak{B}\to\mathfrak{B})\to\mathfrak{B}}$ and $e^{(\mathfrak{B}\to\mathfrak{B})\to\mathfrak{B}}$. The lambda term

$$(\lambda f^{(\mathfrak{B}\to\mathfrak{B})\to\mathfrak{B}}.\lambda e^{(\mathfrak{B}\to\mathfrak{B})\to\mathfrak{B}}.\phi^*)(F,E)$$

where the terms F and E are the same as in the construction above:

$$F = \lambda g^{\mathfrak{B}\to\mathfrak{B}}.and[g(true), g(false)]$$
$$E = \lambda g^{\mathfrak{B}\to\mathfrak{B}}.or[g(true), g(false)]$$

is of the fifth order and reduces to *true* or *false* according to the validity of the formula ϕ. Thus we have:

Fact 7. *Testing equality of simply typed lambda terms of order not greater than five is PSPACE-hard.*

(This is essentially Proposition 6 from [21], page 81). This explains why we have an impression that there is a kind of cheating in our PSPACE-hardness bound for matching: instead of showing the hardness of matching itself, we focused on complexity of β-reduction. So in the next section we refine the previous translation in order to make nontrivial use of unknown variables in a matching problem: we are going to code Schönfinkel-Bernays formulas and test their satisfiability over a finite domain.

4 Fourth Order Matching is NEXPTIME-Hard

Given an infinite set of individual variables $p, q, \ldots$ a collection of constants $c, d, \ldots$ and predicate letters $P, Q, \ldots$ of fixed arity. An atomic formula is of the form $P(t_1, \ldots, t_k)$, where t_i are either constants or variables, and P is a k-ary predicate letter. *Predicate formulas* are built from atomic formulas using negation symbol $\neg$ and binary connectives $\vee$ and $\wedge$ with quantification $(\forall, \exists)$ over individual variables. Sometimes one also adds function symbols allowing building more complicated atomic formulas, and equality symbol. Although the SAT problem for the predicate calculus is undecidable in the general case, some special cases are decidable. A formula ψ is called a Schönfinkel-Bernays formula (sometimes denoted $\exists^*\forall^*$), if it is of the form

$$\psi = \exists p_1 \ldots \exists p_i.\forall q_1 \ldots \forall q_j.\phi$$

that is, it is in prenex form where all existential quantifiers precede all universal quantifiers, and the formula ϕ is built using only constants and predicate letters, with no function symbols and equality. The problem of deciding if such a formula has a model is decidable and NEXPTIME-complete. The key idea in showing decidability of that case follows from the following:

Lemma 1 (Bernays, Schönfinkel [3]). *A Schönfinkel-Bernays formula is satisfiable, if it has a model with exactly n elements, where n is the length of that formula.*

In fact such a formula is either unsatisfiable or has a model with cardinality at most i, where i is the number of existential quantifiers occurring in it. We need the size of a model to be easily calculable for a given formula. In the absence of equality such a formula has models with any greater cardinality, in particular with cardinality n (to increase the size of a model one can pick an arbitrary element of this model and replace it with any number of its copies; in the presence of equality it is impossible, as the sentence "any of my models contains exactly k elements" becomes expressible). Another way of fixing the size of a model is by using relativization. For details see [4,17].

Let $\mathfrak{D}$ abbreviates $o^n \to o$. The type $\mathfrak{D}$ is inhabited by n elements

$$\pi_i^n = \lambda u_1^o. \dots \lambda u_n^o.u_i$$

assumed to represent numbers $i = 1, \dots, n$. It is well known that in this encoding any predicate $B \subseteq \{1, \dots, n\}^k$ is λ-definable as a lambda term $\bar{B} : \mathfrak{D}^k \to \mathfrak{B}$. For example for binary predicate $B(i,j)$, choose two fresh term variables u^o and v^o, let

$$w_{i,j} = \begin{cases} u^o, & \text{when } B(i,j), \\ v^o, & \text{otherwise,} \end{cases}$$

for $i, j \in \{1, \dots, n\}$, and define

$$\begin{aligned}\bar{B} \;=\; \lambda x^{\mathfrak{D}}.\lambda y^{\mathfrak{D}}.\lambda u^o.\lambda v^o.x\Big(\, & y(w_{1,1}, w_{1,2}, \;\dots, w_{1,n}), \\ & y(w_{2,1}, w_{2,2}, \;\dots, w_{2,n}), \\ & \qquad\vdots \\ & y(w_{n,1}, w_{n,2}, \dots, w_{n,n})\Big)\end{aligned}$$

The predicate B is simply tabularized. The lambda term $\bar{B}$ when applied to two values (π_i^n, π_j^n) works as follows: π_i^n (substituted for x), as the i-th selector, picks the i-th row, then π_j^n (substituted for y) picks the j-th element of that row, which happens to be exactly $w_{i,j}$. But $w_{i,j}$ has been defined to be u or v depending on $B(i,j)$, thus $\bar{B}(\pi_i^n, \pi_j^n)$ reduces to $\lambda u^o.\lambda v^o.u = \mathit{true}$, when $B(i,j)$ is true, and to $\lambda u^o.\lambda v^o.v = \mathit{false}$ otherwise.

In particular the equality predicate is definable in our encoding, thus we might enrich the Schönfinkel-Bernays formulas by equality (it is known not to alter the complexity of the SAT problem, however Lemma 1 above is no longer valid). This coding may be extended to arbitrary k-ary predicates. Note that the length of the encoding is $O(n^k)$.

Now we may define quantification over elements of $\mathfrak{D}$ in the same way we did it for $\mathfrak{B}$:

$$\begin{aligned}\mathcal{Q}_{\mathfrak{D}} \;=\; \Big\{ & f^{(\mathfrak{D}\to\mathfrak{B})\to\mathfrak{B}} \overset{?}{=} \lambda g^{\mathfrak{D}\to\mathfrak{B}}.\mathit{and}[\mathit{and}[\dots[\mathit{and}[g(\pi_1^n), g(\pi_2^n)], \dots, g(\pi_n^n)]\dots]], \\ & e^{(\mathfrak{D}\to\mathfrak{B})\to\mathfrak{B}} \overset{?}{=} \lambda g^{\mathfrak{D}\to\mathfrak{B}}.\mathit{or}[\mathit{or}[\dots[\mathit{or}[g(\pi_1^n), g(\pi_2^n)], \dots, g(\pi_n^n)]\dots]]\Big\}\end{aligned}$$

The variables $f^{(\mathfrak{D}\to\mathfrak{B})\to\mathfrak{B}}$ and $e^{(\mathfrak{D}\to\mathfrak{B})\to\mathfrak{B}}$ are of the fourth order. Note that the right hand sides are in normal form, and that the size of $\mathcal{Q}_{\mathfrak{D}}$ is $O(n)$.

Now choosing an n-element set $\mathfrak{D}$ as a possible model of a Schönfinkel-Bernays formula ψ of length n, we may define a translation $\cdot^\star$: for any individual variable p choose a fresh term variable $x_p^{\mathfrak{D}}$, for any constant c choose a fresh term variable $x_c^{\mathfrak{D}}$, for any k-ary predicate letter P choose a fresh variable $x_P^{\mathfrak{D}^k\to\mathfrak{B}}$, and define:

$$\begin{aligned}
p^\star &= x_p^{\mathfrak{D}}\\
c^\star &= x_c^{\mathfrak{D}}\\
(P(t_1,\ldots,t_k))^\star &= x_P^{\mathfrak{D}^k\to\mathfrak{B}}(t_1^\star,\ldots,t_k^\star)\\
(\phi\wedge\psi)^\star &= a^{\mathfrak{B},\mathfrak{B}\to\mathfrak{B}}(\phi^\star,\psi^\star)\\
(\phi\vee\psi)^\star &= r^{\mathfrak{B},\mathfrak{B}\to\mathfrak{B}}(\phi^\star,\psi^\star)\\
(\neg\phi)^\star &= n^{\mathfrak{B}\to\mathfrak{B}}(\phi^\star)\\
(\forall p.\phi)^\star &= f^{(\mathfrak{D}\to\mathfrak{B})\to\mathfrak{B}}(\lambda x_p^{\mathfrak{D}}.\phi^\star)\\
(\exists p.\phi)^\star &= e^{(\mathfrak{D}\to\mathfrak{B})\to\mathfrak{B}}(\lambda x_p^{\mathfrak{D}}.\phi^\star)
\end{aligned}$$

Note we may also code this way the full first order predicate calculus with function symbols and equality. It is however irrelevant, since the main problem is the size of a model we are able to build for checking satisfiability, not the expressive power of the language.

Given a Schönfinkel-Bernays formula ψ. Clearly the fourth order matching problem $\mathcal{B}\cup\mathcal{Q}_{\mathfrak{D}}\cup\{\psi^\star \overset{?}{=} \mathit{true}\}$ (which may be constructed in time polynomial in the size of ψ) has a solution when ψ is satisfiable (*i.e.*, by Lemma 1 when it has a model of cardinality n; thus code its model with n elements in the lambda calculus in the way described above; the key point is that every predicate is λ-definable). From the other hand if it has a solution, then its solution is immediately a model for ψ (the key point in this direction is that there is no lambda term of type $\mathfrak{D}$ other than π_i^n for $i=1,\ldots,n$). Thus, since the SAT problem for ψ is NEXPTIME-hard we have proved the following:

Fact 8. *Fourth order pure matching is NEXPTIME-hard.*

5 Final Remarks

The status of higher order matching is known for the order up to four. The fifth order case is not even known to be decidable, however some partial results have already been obtained [19]. Even if Huet's conjecture stating the decidability of matching in the general case is true [11], it is of no importance for practical applications, since complexity of higher order cases grows very quickly, and from the practical point of view problems that are NEXPTIME-hard seem to be as difficult as undecidable ones. We do not suggest however that higher order matching is not tractable, *e.g.*, Huet's semidecision procedure has been shown to be useful in, say, theorem provers even though unification is in general undecidable. It turns out that undecidability is not an obstacle in solving such problems in practice! Our result implies only that there is no hope to find a fast, sound and at the same time *complete* algorithm for higher order matching.[4]

Acknowledgments

I would like to thank anonymous referees for their critical comments and Oskar Miś for long and interesting discussions.

References

1. Henk Barendregt, Lambda Calculi with Types, *Handbook of Logic in Comput. Sci.*, vol. 2, S. Abramsky, D.M. Gabbay, T.S.E. Maibaum, eds., Clarendon Press, Oxford, 1992, 118–310.
2. Lewis D. Baxter, *The Complexity of Unification*, Ph.D. Thesis, University of Waterloo, 1976.
3. Paul Bernays, Moses Schönfinkel, Zum Entscheidungsproblem der mathematischen Logik, *Math. Annalen*, **99** (1928) 342–372.
4. Egon Börger, Erich Grädel, Yuri Gurevich, *The Classical Decision Problem*, Springer-Verlag, 1997.
5. Hubert Comon, Yan Jurski, Higher order pattern matching and tree automata, *Proc. 11th Int'l Workshop on Comput. Sci. Logic, CSL'97*, Aarhus, Denmark, August 23–29, 1997, M. Nielsen, W. Thomas, eds., LNCS **1414**, Springer-Verlag, 1998.
6. Gilles Dowek, Third order matching is decidable, *Proc. 7th IEEE Symp. Logic in Comput. Sci., LICS'92*, Santa Cruz, California, June 22–25, 1992, 2–10, also in *Annals of Pure and Applied Logic*, **69** (1994), 135–155.

[4] One of the referees suggested that there is no common agreement about reasons why complexity theory sometimes diverges from practice, and that this statement is not sufficiently justified. I would like to stress that this is my personal opinion which follows from my experience with another problem also theoretically proved to be hard and at the same time found to be efficiently solvable in practice, *i.e.*, with ML type system. ML typability (decision problem) as well as type reconstruction (a task of explicitly writing down a type of a program) are both DEXPTIME-hard, while in practice type reconstruction turns out to be solvable in nearly linear time (real-life programs consist sometimes of $n > 10^5$ abstract syntax tree nodes for which type reconstruction is invoked; even procedure running in $O(n^2)$ time would be impractical). This is because types in real-life programs must be very short to be meaningful for a programmer, and practical experiments show, that they are in fact trivial: their average size is less than 6 no matter how large a program is, while theory allows types of dag size $2^{O(n)}$. However any ML compiler fails to compile a five-line artificial program whose type is of exponential dag size. Probably the status of matching is analogous: meaningful theorems and their proofs should be short enough to be understood by humans and in practical applications of theorem provers the whole complexity of matching (or unification) is not exploited.

7. Gilles Dowek, The undecidability of pattern matching in calculi where primitive recursive functionals are representable, *Theoret. Comput. Sci.*, **107** (1993) 349–56.
8. Cynthia Dwork, Paris C. Kanellakis, John C. Mitchell, On the sequential nature of unification, *J. Logic Programming*, **1** (1) (1984) 35–50.
9. Warren D. Goldfarb, Note on the undecidability of the second order unification problem, *Theoret. Comput. Sci.* **13** (1981) 225–230.
10. Gerard P. Huet, A unification algorithm for typed λ-calculus, *Theoret. Comput. Sci.*, **1** (1) (1975) 27–57.
11. Gerard P. Huet, *Résolution d'équations dans des langages d'ordre* $1, 2, \ldots, \omega$. Thèse de Doctorat d'État, University of Paris, 1976.
12. Gerard P. Huet, Bernard Lang, Proving and applying program transformations expressed with second order patterns, *Acta Informatica*, **11** (1978) 31–55.
13. Ralph Loader, The undecidability of λ-definability, *Church Memorial Volume*, A. Anderson, M. Zeleny eds., Kluwer Acad. Press, to appear.
14. Harry G. Mairson, A simple proof of a theorem of Statman, *Theoret. Comput. Sci.*, **103** (1992) 387–394.
15. Dale Miller, A logic programming language with lambda-abstraction, function variables, and simple unification, *J. Logic and Comput.*, **1** (4) (1991) 497–536.
16. Vincent Padovani, *Filtrage d'ordre supérieur*, PhD Thesis, Université Paris VII, 1996.
17. Christos H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.
18. John Alan Robinson, A machine-oriented logic based on the resolution principle, *J. ACM*, **12** (1) (1965) 23–41.
19. Aleksy Schubert, Linear interpolation for the higher order matching problem, *Technical Report of the Institute of Informatics, Warsaw University*, TR 96-16 (237), also in *Proc. 7th Int'l Joint Conf. Theory and Practice of Software Development, TAPSOFT'97*, Lille, France, April 14–18, 1997, M. Bidoit, M. Dauchet, eds., LNCS **1214**, Springer-Verlag, 1997.
20. Richard Statman, Intuitionistic propositional logic is polynomial-space complete, *Theoret. Comput. Sci.*, **9** (1979) 67–72.
21. Richard Statman, The typed λ-calculus is not elementary recursive, *Theoret. Comput. Sci.*, **9** (1979) 73–81.
22. Richard Statman, Completeness, invariance and λ-definability, *J. Symbolic Logic*, **47** (1) (1982) 17–26.
23. Richard Statman, On the existence of closed terms in the typed λ-calculus II: transformations of unification problems, *Theoret. Comput. Sci.*, **15** (1981) 329–338.
24. Sergei Vorobyov, The "Hardest" Natural Decidable Theory, *Proc. 12th Annual IEEE Symp. Logic in Comput. Sci., LICS'97*, Warsaw, Poland, June 29 – July 2, 1997, 294–305.
25. David A. Wolfram, The decidability of higher-order matching, *Proc. 6th Int'l Workshop on Unification*, Schloß Dagstuhl, Germany, July 29–31, 1992.
26. David A. Wolfram, *The Clausal Theory of Types*, Cambridge Tracts in Theor. Comput. Sci. vol. 36, Cambridge University Press, 1993.

Solving Equational Problems Efficiently

Reinhard Pichler

Technische Universität Wien
reini@logic.at

Abstract. Equational problems (i.e.: first-order formulae with quantifier prefix $\exists^* \forall^*$, whose only predicate symbol is syntactic equality) are an important tool in many areas of automated deduction, e.g.: restricting the set of ground instances of a clause via equational constraints allows the definition of stronger redundancy criteria and hence, in general, of more efficient theorem provers. Moreover, also the inference rules themselves can be restricted via constraints. In automated model building, equational problems play an important role both in the definition of an appropriate model representation and in the evaluation of clauses in such models. Also, many problems in the area of logic programming can be reduced to equational problem solving.
The goal of this work is a complexity analysis of the satisfiability problem of equational problems in CNF over an infinite Herbrand universe. The main result will be a proof of the NP-completeness (and, in particular, of the NP-membership) of this problem.

1 Introduction

Equational problems (i.e.: first-order formulae with quantifier prefix $\exists^* \forall^*$, whose only predicate symbol is syntactic equality) are an important tool in many areas of automated deduction:

Restricting the set of ground instances of a clause via equational constraints allows the definition of stronger redundancy criteria and hence, in general, of more efficient theorem provers: In [5], the c-distautology rule allows the deletion of those ground instances of a clause, which are tautological. To this end, the instances of a clause $L(\boldsymbol{s}\,) \vee L^d(\boldsymbol{t}\,) \vee R$ are restricted via the constraint $\boldsymbol{s} \neq \boldsymbol{t}$. Likewise, the c-dissubsumption rule allows the deletion of those ground instances which are subsumed by another clause. Moreover, in [4], semantic c-resolution is defined, where the inference rules themselves are restricted via equational constraints.

In automated model building, equational problems play a crucial role, e.g.: In [5], models are represented by unit clauses whose set of ground instances may be restricted via equational constraints. In [6], models are represented by atom sets $\mathcal{A} = \{P_1(\boldsymbol{t}_1), \ldots, P_n(\boldsymbol{t}_n)\}$ with the intended meaning that a ground atom $P(\boldsymbol{s}\,)$ evaluates to true, iff it is an instance of some atom $P_i(\boldsymbol{t}_i) \in \mathcal{A}$. The evaluation of an arbitrary atom to false in such a model corresponds to the satisfiability of the equational problem $\mathcal{P} \equiv (\exists \boldsymbol{x}\,) \bigwedge_{P_i = P} (\forall \boldsymbol{y}_i)(\boldsymbol{s} \neq \boldsymbol{t}_i)$, where the variables $\boldsymbol{x}$ in

$P(\boldsymbol{s})$ and the variables $\boldsymbol{y}_i$ in $P_i(\boldsymbol{t}_i)$ are disjoint. In [8], implicit generalizations are studied as a formal basis of machine learning from counter-examples. For testing the emptiness of implicit generalizations, equational problems similar to the above mentioned evaluation of atoms arise. Likewise, in functional programming, the problem of checking whether a function defined by case is completely defined can be reduced to equational problems of this kind (cf. [9]).

In the area of logic programming, equational problems may be applied in various ways. In particular, the definition of an appropriate semantics of negation can be reduced to equational problem solving (cf. [8], [10], [15]).

In most of the above mentioned applications of equational problems, testing the satisfiability of an equational problem is more important than actually computing the solutions. Hence, for the practical work with equational problems, an efficient satisfiability test is a central issue. All previously known algorithms require exponential time and space. The goal of this work is an investigation of the inherent complexity of this satisfiability problem. Note that even in the special case of a purely existential prefix, this problem is NP-hard. Our main result will be a proof of the NP-membership (and, therefore, of the NP-completeness) for equational problems in CNF over an infinite Herbrand universe.

This work is organized as follows: After briefly revising some basic definitions in Section 2, we shall illustrate in Section 3 what the main sources of complexity of equational problems in CNF are. At the heart of our method for solving equational problems is the elimination of universally quantified variables from the equations. The transformation given in Section 4 for this purpose will form the basis of the NP-membership proof (and, therefore, of the NP-completeness proof) presented in Section 5. Finally, in Section 6 we shall summarize the main results of this work and point out some directions for future research.

2 Preliminaries

2.1 Equational Problems

Equational problems are formulae of the form $\exists \boldsymbol{w} \forall \boldsymbol{y} \mathcal{P}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y})$, s.t. $\mathcal{P}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y})$ is a quantifier-free formula with equality "=" as the only predicate symbol. A disequation $s \neq t$ is a short-hand notation for a negated equation $\neg(s = t)$. The trivially true problem is denoted by $\top$ and the trivially false one by $\bot$.

In this paper, equational problems are only interpreted over the free term algebra. Furthermore, variables of a single sort are considered, i.e.: they may only take values from the same Herbrand universe H. A Herbrand interpretation over H is given through an H-ground substitution σ, whose domain coincides with the free variables of the equational problem. The trivial problem $\top$ evaluates to true in every interpretation. Likewise, $\bot$ always evaluates to false. A single equation $s = t$ is validated by a ground substitution σ, if $s\sigma$ and $t\sigma$ are syntactically identical. The connectives $\wedge$, $\vee$, $\neg$, $\exists$ and $\forall$ are interpreted as usual. A ground substitution σ which validates a problem $\mathcal{P}$ is called a *solution* of $\mathcal{P}$.

In order to distinguish between syntactical identity and the equivalence of two equational problems, we shall use the notation "$\equiv$" and "$\approx$", respectively, i.e.: $\mathcal{P} \equiv \mathcal{Q}$ means that the two equational problems $\mathcal{P}$ and $\mathcal{Q}$ are syntactically identical, while $\mathcal{P} \approx \mathcal{Q}$ means that the two problems are equivalent (i.e.: they have the same set of solutions). Moreover, by $\mathcal{P} \leq \mathcal{Q}$ we denote that all solutions of $\mathcal{P}$ are also solutions of $\mathcal{Q}$.

As far as the satisfiability of an equational problem is concerned, there is no difference between free variables and existentially quantified ones, i.e.: $\exists \boldsymbol{w} \forall \boldsymbol{y} \mathcal{P}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y})$ is satisfiable, iff $\exists \boldsymbol{x} \exists \boldsymbol{w} \forall \boldsymbol{y} \mathcal{P}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y})$ is. In analogy with [3], we shall refer to universally quantified variables as *"parameters"*. Furthermore, the basic form of equational problems studied in [3] is the CNF (= conjunctive normal form). Hence, also the complexity result obtained here will be based on the following form:

Definition 2.1. ($\exists^* \forall^*$-CNF). *An equational problem $\mathcal{P}$ is said to be in $\exists^* \forall^*$-CNF, iff it is of the form $\mathcal{P} \equiv \exists \boldsymbol{z} \forall \boldsymbol{y} \mathcal{P}(\boldsymbol{z}, \boldsymbol{y})$, where $\mathcal{P}(\boldsymbol{z}, \boldsymbol{y})$ is a quantifier-free problem in CNF.*

Unless explicitly stated otherwise, we shall only consider the case of an infinite Herbrand universe here. Finally we shall sometimes use term tuples as a shorthand notation for a disjunction of disequations or a conjunction of equations, respectively, i.e.: For term tuples $\boldsymbol{s} = (s_1, \ldots, s_k)$ and $\boldsymbol{t} = (t_1, \ldots, t_k)$, we shall abbreviate "$s_1 = t_1 \wedge \ldots \wedge s_k = t_k$" and "$s_1 \neq t_1 \vee \ldots \vee s_k \neq t_k$" to "$\boldsymbol{s} = \boldsymbol{t}$" and "$\boldsymbol{s} \neq \boldsymbol{t}$", respectively. Moreover, we shall use a vector $\boldsymbol{x} = (x_1, \ldots, x_k)$ of variables either to denote a tuple of variables (which can be used as an argument of an equation or disequation in the way described above) or to denote a set of variables. No ambiguities will arise from this, since the meaning will always be clear from the context.

2.2 The Transformation Rules of Comon/Lescanne

In [3], a rule system is provided which terminates on every equational problem and which transforms the original problem into an equivalent one in the so-called "definition with constraints form", which allows to determine immediately the satisfiability. Below, those rules from [3] are recalled, which are cited frequently in this paper, i.e.: the replacement rules R_1, R_2, the universality of parameter rules U_2, U_4 and the explosion rule E. Note that many more rules from [3] (like the decomposition rule, the clash rule, the occur check, etc.), which are not mentioned explicitly here, are "hidden" in the unification steps. For any details, the original paper has to be referred to. Analogously to [3], we use "$\rightarrow$", to denote a rule which transforms an equational problem into an equivalent one, and "$\rightharpoonup$" if a combination of several instances of the same rule are required in order to preserve the set of solutions.

(R_1) $\quad z = t \wedge P \quad \rightarrow \quad z = t \wedge P(z \leftarrow t)$

(R_2) $\quad z \neq t \vee P \quad \rightarrow \quad z \neq t \vee P(z \leftarrow t)$

(U_2) $\quad (\forall \boldsymbol{y})[P \wedge (y \neq t \vee R)] \quad \rightarrow \quad (\forall \boldsymbol{y})[P \wedge R(y \leftarrow t)] \quad$ if $y \in \boldsymbol{y}$

(U_4) $(\forall \boldsymbol{y})[P \wedge (z_1 = u_1 \vee \ldots \vee z_n = u_n \vee R)] \quad \rightarrow \quad (\forall \boldsymbol{y})[P \wedge R]$
if the following conditions hold:
1. every z_i is a variable syntactically different from u_i,
2. every equation $z_i = u_i$ contains at least one parameter from $\boldsymbol{y}$,
3. R contains no parameter from $\boldsymbol{y}$.

(E) $(\forall \boldsymbol{y})P \quad \rightharpoonup \quad \exists(w_1, \ldots, w_\alpha)(\forall \boldsymbol{y})[P \wedge s = f(w_1, \ldots, w_\alpha)]$
if the following conditions hold:
1. f is a symbol from the signature of H with arity α,
2. the w_i's are fresh, pairwise distinct variables,
3. s is an argument of an equation or disequation in P and s contains no parameter.

2.3 Unification Problems

If an equational problem is a parameter-free conjunction of equations, then we are back to the familiar case of *unification problems.* In [1], the efficiency of several unification algorithms is analysed. In particular, it is shown, that the original unification algorithm from [14], where terms are represented as strings of symbols, has exponential time and space complexity. However, by using more sophisticated data structures like directed acyclic graphs, this kind of exponential blow-up can be avoided. In fact, even linear time suffices (cf. [12]).

3 Main Sources of Complexity

In this section, we have a closer look at the complexity of two special cases of equational problems, namely: equational problems with only existentially quantified variables and equational problems consisting of disequations only. In fact, the satisfiability problem is NP-complete in both cases. The NP-hardness illustrates that the two main sources of complexity of equational problems in CNF are the total number of equations and disequations, on the one hand, and the number of conjuncts containing disequations with universally quantified variables, on the other hand. The NP-membership of these problems will give us a hint for the NP-membership proof of equational problems in $\exists^* \forall^*$-CNF.

The first goal of the algorithm in [3] is the elimination of all universally quantified variables. Together with the restriction to CNF, we get the following form:

Definition 3.1. (Parameterless CNF). *An equational problem $\mathcal{P}$ is said to be in* parameterless CNF*, iff it is in CNF and contains no universally quantified variables, i.e. $\mathcal{P} \equiv (\exists \boldsymbol{x})[(e_{11} \vee \ldots \vee e_{1k_1} \vee d_{11} \vee \ldots \vee d_{1l_1}) \wedge \ldots \wedge (e_{n1} \vee \ldots \vee e_{nk_n} \vee d_{n1} \vee \ldots \vee d_{nl_n})]$, s.t. the e_{ij}'s are equations and the d_{ij}'s are disequations.*

Testing the satisfiability of equational problems in parameterless CNF can be easily shown to be NP-complete: For the *NP-hardness proof*, the problem reduction from the SAT-problem is straightforward, namely: Choose two arbitrary, distinct terms s and t from H, s.t. "true" is encoded by s and "false" is encoded by t. Then, for every propositional variable x_i in SAT, a clause of the form $x_i = s \vee x_i = t$ is required in the equational problem. Moreover, for every positive occurrence of x_i in a clause in SAT, an equation $x_i = s$ is contained in the corresponding clause in the equational problem. Likewise, a negative occurrence of x_i in SAT is encoded by the equation $x_i = t$. At the heart of the *NP-membership proof* is the polynomial time test for the satisfiability of parameterless conjunctions of equations and disequations given in Lemma 3.1 below. This test will also play an essential role in Section 5 when we prove the NP-membership of equational problems in $\exists^* \forall^*$-CNF.

Lemma 3.1. (Parameterless Conjunctions). *Let $\mathcal{P} \equiv (\exists \boldsymbol{x})[e_1 \wedge \ldots \wedge e_k \wedge d_1 \wedge \ldots \wedge d_l]$ be a conjunction of equations e_i and disequations d_j. Then the satisfiability of $\mathcal{P}$ can be tested as follows:*

- *case 1: If $e_1 \wedge \ldots \wedge e_k$ is unsatisfiable, then $\mathcal{P}$ is also unsatisfiable.*
- *case 2: Suppose that $e_1 \wedge \ldots \wedge e_k$ is satisfiable with* m*gu* ϑ. *Then $\mathcal{P}$ is satisfiable, iff $d_1\vartheta \wedge \ldots \wedge d_l\vartheta$ contains no trivial disequation of the form $t \neq t$.*

Proof: Case 1 is trivial. For case 2, let $\vartheta = \{x_{i_1} \leftarrow s_1, \ldots, x_{i_\alpha} \leftarrow s_\alpha\}$ denote the m*gu* of the equations $e_1 \wedge \ldots \wedge e_k$. Note that the x_{i_j} are pairwise distinct and do not occur in the range r$g(\vartheta)$ of ϑ.

By the definition of the m*gu*, $e_1 \wedge \ldots \wedge e_k$ and $x_{i_1} = s_1 \wedge \ldots \wedge x_{i_\alpha} = s_\alpha$ are equivalent. Moreover, by multiple applications of the R_2-rule from [3], ϑ may be applied to the disequations. Thus $\mathcal{P} \approx (\exists \boldsymbol{x})[x_{i_1} = s_1 \wedge \ldots \wedge x_{i_\alpha} = s_\alpha \wedge d_1\vartheta \wedge \ldots \wedge d_l\vartheta]$ holds. But then, since all variables $x_{i_1}, \ldots, x_{i_\alpha}$ occur only once, the equations may be eliminated by the CR_2-rule from [3], i.e.: $\mathcal{P} \approx \mathcal{P}' \equiv (\exists \boldsymbol{x})[d_1\vartheta \wedge \ldots \wedge d_l\vartheta]$. By Lemma 1 in [3], a conjunction of non-trivial disequations over an infinite Herbrand universe has at least one solution. But then $\mathcal{P}'$ (and, hence, also $\mathcal{P}$) is satisfiable, iff $\mathcal{P}'$ contains no trivial disequation. $\diamond$

We thus get the following theorem:

Theorem 3.1. (NP-Completeness of Parameterless CNF). *The satisfiability problem for equational problems in parameterless CNF is NP-complete.*

In [13], an efficient algorithm for solving the so-called TTC problem (= *term tuple cover problem*) is presented, i.e.: Given a set $M = \{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n\}$ of k-tuples of terms over some Herbrand universe H. Is every ground term tuple $\boldsymbol{s} \in H^k$ an instance of some tuple $\boldsymbol{t}_i \in M$? In [7], the coNP-completeness of the term tuple cover problem is proven. The complementary problem coTTC can be easily translated into the satisfiability problem of equational problems whose form is captured by Definition 3.2 below. But then Theorem 3.2 on the complexity of equational problems follows immediately.

Definition 3.2. (coTTC Form). *An equational problem $\mathcal{P}$ is said to be in* coTTC form, *iff it is of the form $\mathcal{P} \equiv (\exists \boldsymbol{x})[(\forall \boldsymbol{y}_1)(\boldsymbol{x} \neq \boldsymbol{t}_1) \wedge \ldots \wedge (\forall \boldsymbol{y}_n)(\boldsymbol{x} \neq \boldsymbol{t}_n)]$, s.t. for all $i \in \{1, \ldots, n\}$, $\boldsymbol{t}_i = (t_{i1}, \ldots, t_{ik})$ is a term tuple with variables in $\boldsymbol{y}_i$.*

Theorem 3.2. (NP-Completeness of coTTC Form). *The satisfiability problem for equational problems in coTTC form is NP-complete.*

4 Parameter-Free Equations

In the previous section, the total number of disjuncts in the CNF and the number of disequations containing parameters have been identified as main sources of complexity of equational problems. On the other hand, the parameters contained in the equations can be shown to have a much lower impact on the overall complexity. In fact, we shall provide in this section a polynomial time transformation for eliminating all parameters from the equations. Moreover, as a by-product of this transformation, the form of the disequations will be simplified. The target of this section is, therefore, a transformation of equational problems into the following form (which is similar to the normal form presented in [11]):

Definition 4.1. (PFE-Form). *An equational problem $\mathcal{P}$ is said to be in PFE-form (=* parameter-free equations*), iff it has the form*

$$(\exists \boldsymbol{x})[(e_{11} \vee \ldots \vee e_{1k_1}) \vee (\forall \boldsymbol{y}_1)(\boldsymbol{z}_1 \neq \boldsymbol{t}_1)] \wedge \ldots \wedge [(e_{n1} \vee \ldots \vee e_{nk_n}) \vee (\forall \boldsymbol{y}_n)(\boldsymbol{z}_n \neq \boldsymbol{t}_n)],$$

s.t. the e_{ij}'s are equations, every $\boldsymbol{t}_i$ is a term tuple with variables in $\boldsymbol{y}_i$ and every $\boldsymbol{z}_i$ denotes a tuple of existentially quantified variables from $\boldsymbol{x}$.

Before we can start with the elimination of the parameters from the equations, we first have to simplify the disequations via the following lemma:

Lemma 4.1. (Simplification of Disequations). *Let $\mathcal{P} \equiv (\forall \boldsymbol{y})[\mathcal{R} \vee \boldsymbol{s} \neq \boldsymbol{t}]$ be an equational problem with free variables in $\boldsymbol{x}$, where $\mathcal{R}$ denotes an arbitrary equational problem. Then $\mathcal{P}$ can be simplified via unification in the following way:*

- *case 1: If the equation $\boldsymbol{s} = \boldsymbol{t}$ is not unifiable, then $\mathcal{P} \approx \top$.*
- *case 2: Suppose that $\vartheta = \{x_{i_1} \leftarrow u_1, \ldots, x_{i_k} \leftarrow u_k, y_{j_1} \leftarrow v_1, \ldots, y_{j_l} \leftarrow v_l\}$ is the mgu of $\boldsymbol{s} = \boldsymbol{t}$ and let the substitution η be defined as $\eta = \{y_{j_1} \leftarrow v_1, \ldots, y_{j_l} \leftarrow v_l\}$. Then $\mathcal{P}$ is equivalent to $\mathcal{P}' \equiv (\forall \boldsymbol{y})[\mathcal{R}\eta \vee x_{i_1} \neq u_1 \vee \ldots \vee x_{i_k} \neq u_k]$.*

Proof: In case 1, there exists no substitution σ on the variables $\boldsymbol{x} \cup \boldsymbol{y}$ s.t. $\boldsymbol{s}\sigma = \boldsymbol{t}\sigma$. Hence, $\boldsymbol{s} \neq \boldsymbol{t} \approx \top$ and also $\mathcal{P} \approx \top$ clearly hold.

In case 2, the equivalence $\boldsymbol{s} \neq \boldsymbol{t} \approx x_{i_1} \neq u_1 \vee \ldots \vee x_{i_k} \neq u_k \vee y_{j_1} \neq v_1 \vee \ldots \vee y_{j_l} \neq v_l$ holds by the definition of the m*gu*. Note that the variables y_{j_α} occur exactly once in the disequations thus produced. Hence, l applications of the U_2-rule from [3] to the disequations $y_{j_1} \neq v_1, \ldots, y_{j_l} \neq v_l$ can be contracted to a single transformation step via the substitution η, i.e.: $\mathcal{P} \approx (\forall \boldsymbol{y})[\mathcal{R}\eta \vee (x_{i_1} \neq u_1 \vee \ldots \vee x_{i_k} \neq u_k)\eta]$. Finally, the substitution η only needs to be applied to $\mathcal{R}$, since the variables y_{j_α} from the domain of η do not occur in the disequations $x_{i_\beta} \neq u_\beta$. ◇

The U_4-rule from [3] allows the elimination of those parameters from the equations which do not occur in the disequations. However, the U_4-rule is very restrictive in that it requires the equations to be of the form $z = u$, where z is a variable. In the following lemma we show how all parameters that do not occur in the disequations can be eliminated from the equations.

Lemma 4.2. (Elimination of Parameters not Occurring in the Disequations). *Let $\mathcal{P} \equiv (\forall \boldsymbol{y})(s_1 = t_1 \vee \ldots \vee s_m = t_m \vee \mathcal{R})$ be an equational problem with free variables in $\boldsymbol{x}$, s.t. every equation $s_i = t_i$ contains at least one parameter from $\boldsymbol{y}$ and $\mathcal{R}$ contains no parameter from $\boldsymbol{y} = \{y_1, \ldots, y_k\}$. Then the parameters $\boldsymbol{y}$ can be eliminated from $\mathcal{P}$ by the following rules:*

1. *non-unifiable equations: If an equation $s_i = t_i$ is not unifiable, then $s_i = t_i$ may be deleted from $\mathcal{P}$.*
2. *parameters not occurring in the mgu: If $s_i = t_i$ is unifiable with mgu ϑ and no parameter y_α occurs in ϑ (i.e.: $\mathrm{dom}(\vartheta) \cap \boldsymbol{y} = \emptyset$ and $\mathrm{Var}(\mathrm{rg}(\vartheta)) \cap \boldsymbol{y} = \emptyset$), then every occurrence of every parameter $y_\alpha \in \boldsymbol{y}$ in $s_i = t_i$ may be replaced by an arbitrary constant symbol $a \in H$.*
3. *parameters occurring in the mgu: If $s_i = t_i$ is unifiable with mgu ϑ and at least one parameter y_α occurs in ϑ (i.e.: $\mathrm{dom}(\vartheta) \cap \boldsymbol{y} \neq \emptyset$ or $\mathrm{Var}(\mathrm{rg}(\vartheta)) \cap \boldsymbol{y} \neq \emptyset$), then the equation $s_i = t_i$ may be deleted from $\mathcal{P}$.*

Proof: Case 1 is obvious. Case 2 is also clear, since the mgu ϑ completely describes all solutions of an equation and replacing variables from outside the mgu does not change the mgu. Hence, we are only left with case 3: Let I denote the indices of the equations to which case 3 applies. Note that by the cases 1 and 2, all parameters can be eliminated from the equations $s_j = t_j$ for $j \notin I$. Hence, $\mathcal{P}$ is equivalent to $\mathcal{P}' \equiv (\forall \boldsymbol{y}) \bigvee_{i \in I}(s_i = t_i) \vee \mathcal{R}'$ for appropriately chosen $\mathcal{R}'$ s.t. $\mathcal{R}'$ contains no parameter from $\boldsymbol{y}$. For every $i \in I$ let $\vartheta_i = \{z_i \leftarrow u_i\} \cup \eta_i$ denote the mgu of $s_i = t_i$ s.t. some parameter $y_{\alpha_i} \in \boldsymbol{y}$ occurs in z_i or u_i. Then, for $\mathcal{P}'' \equiv (\forall \boldsymbol{y}) \bigvee_{i \in I}(z_i = u_i) \vee \mathcal{R}'$, the relation $\mathcal{P}' \leq \mathcal{P}''$ holds, since the omission of parts of the variable bindings in the mgu may not decrease the set of solutions. But then the equations $z_i = u_i$ may be deleted by the U_4-rule from [3]. Thus, $\mathcal{P}''$ is equivalent to $(\forall \boldsymbol{y})\mathcal{R}'$. Hence, the equivalence $\mathcal{P} \approx (\forall \boldsymbol{y})\mathcal{R}'$ follows from the relations $\mathcal{P} \approx \mathcal{P}' \leq \mathcal{P}'' \approx (\forall \boldsymbol{y})\mathcal{R}' \leq \mathcal{P}$. ◇

The following lemma allows the elimination of the remaining parameters from the equations. Note that again there is no appropriate transformation rule from [3] available for this purpose: Eliminating these parameters via the explosion rule has exponential time complexity. And the U_2-rule can only be used to eliminate the parameters from the equations if all disequations $x_{i_j} \neq t_j$ have been broken down to disequations between variables only. Again the necessary applications of the explosion rule require exponential time.

Lemma 4.3. (Elimination of the Remaining Parameters from the Equations). *Let $\mathcal{P} \equiv (\forall \boldsymbol{y})(\mathcal{E} \vee \boldsymbol{z} \neq \boldsymbol{t})$ be an equational problem with free variables in $\boldsymbol{x}$, s.t. $\mathcal{E} \equiv s_1 = t_1 \vee \ldots \vee s_k = t_k$ is a disjunction of equations, $\boldsymbol{z} \subseteq \boldsymbol{x}$ is a vector of free variables which do not occur in the term tuple $\boldsymbol{t}$ and all parameters in $\boldsymbol{y}$ actually do occur in $\boldsymbol{t}$. Then $\mathcal{P}$ is equivalent to $\mathcal{P}' \equiv [(\exists \boldsymbol{y})(\boldsymbol{z} = \boldsymbol{t} \wedge \mathcal{E})] \vee [(\forall \boldsymbol{y})\boldsymbol{z} \neq \boldsymbol{t}]$.*

Proof: By assumption, the variables in $\boldsymbol{z}$ occur only once in the disequations. Hence, the equivalence $\mathcal{P} \approx (\forall \boldsymbol{y})[(\boldsymbol{z} = \boldsymbol{t} \wedge \mathcal{E}) \vee \boldsymbol{z} \neq \boldsymbol{t}]$ basically corresponds to multiple applications of the R_2-rule from [3]. (Note that in contrast to the original form of the R_2-rule, we only add the equation $\boldsymbol{z} = \boldsymbol{t}$ to the first disjunct rather than applying the corresponding substitutions $\{z_1 \leftarrow t_1\}, \{z_2 \leftarrow t_2\}$, etc. to the variables in $\mathcal{E}$. But this is clearly equivalent to the R_2-rule itself). Replacing the universal quantifier for some variables by an existential one cannot decrease the set of solutions. Hence, the relation $\mathcal{P} \leq \mathcal{P}'$ clearly holds. It therefore only remains to show that every solution of $\mathcal{P}'$ is also a solution of $\mathcal{P}$. Let the ground substitution σ on the free variables $\boldsymbol{x}$ be a solution of $\mathcal{P}'$. If σ is a solution of $(\forall \boldsymbol{y})\boldsymbol{z} \neq \boldsymbol{t}$, then it is of course also a solution of $\mathcal{P} \equiv (\forall \boldsymbol{y})(\mathcal{E} \vee \boldsymbol{z} \neq \boldsymbol{t})$. So suppose that σ is a solution of $(\exists \boldsymbol{y})(\boldsymbol{z} = \boldsymbol{t} \wedge \mathcal{E})$ and let τ be an arbitrary ground substitution for the variables $\boldsymbol{y}$. We have to show that then $(\mathcal{E} \vee \boldsymbol{z} \neq \boldsymbol{t})(\sigma \cup \tau) \approx \top$ holds:

If $(\boldsymbol{z} \neq \boldsymbol{t})(\sigma \cup \tau) \approx \top$ holds, then $(\mathcal{E} \vee \boldsymbol{z} \neq \boldsymbol{t})(\sigma \cup \tau) \approx \top$ clearly holds. Hence, we only have to prove the equivalence $\mathcal{E}(\sigma \cup \tau) \approx \top$ for the case where $(\boldsymbol{z} \neq \boldsymbol{t})(\sigma \cup \tau) \approx \perp$ holds:

By assumption, σ is a solution of $(\exists \boldsymbol{y})(\boldsymbol{z} = \boldsymbol{t} \wedge \mathcal{E})$. Therefore, there exists a ground substitution φ for the variables $\boldsymbol{y}$ s.t. $(\boldsymbol{z} = \boldsymbol{t} \wedge \mathcal{E})(\sigma \cup \varphi) \approx \top$. In particular, $\mathcal{E}(\sigma \cup \varphi) \approx \top$ holds. It is, therefore, sufficient to show that τ and φ are identical: Remember that we are dealing with the case where $(\boldsymbol{z} \neq \boldsymbol{t})(\sigma \cup \tau) \approx \perp$ and, therefore, $(\boldsymbol{z} = \boldsymbol{t})(\sigma \cup \tau) \approx \top$ holds, i.e.: $\sigma \cup \tau$ is a unifier of $\boldsymbol{z} = \boldsymbol{t}$. Furthermore, $(\boldsymbol{z} = \boldsymbol{t} \wedge \mathcal{E})(\sigma \cup \varphi) \approx \top$ holds by the definition of φ and, therefore, $\sigma \cup \varphi$ is also a unifier of $\boldsymbol{z} = \boldsymbol{t}$. By assumption, all parameters y_i actually do occur in $\boldsymbol{t}$. Hence, for every unifier $(\sigma \cup \eta)$ of $\boldsymbol{z} = \boldsymbol{t}$, where σ is a ground substitution on the variables $\boldsymbol{x}$, the extension η to the variables $\boldsymbol{y}$ is uniquely determined by σ. But then, φ and τ are indeed identical. ◇

Note that the conjunction $(\boldsymbol{z} = \boldsymbol{t} \wedge \mathcal{E})$ in the above lemma with $\mathcal{E} \equiv s_1 = t_1 \vee \ldots \vee s_k = t_k$ can be represented as a disjunction $\mathcal{E}'$ of equations (of term tuples), namely: $\mathcal{E}' \equiv \big(\boldsymbol{z}, s_1\big) = \big(\boldsymbol{t}, t_1\big) \vee \ldots \vee \big(\boldsymbol{z}, s_k\big) = \big(\boldsymbol{t}, t_k\big)$ Hence, the only part missing in our transformation into PFE-form is the elimination of the existentially quantified variables from the right-hand side of the disequations. But this can be easily achieved by applying the U_2-rule from [3] in the opposite direction:

Lemma 4.4. (Elimination of the Existentially Quantified Variables from the Right-Hand Side of the Disequations). *Let $\mathcal{P} \equiv (\forall \boldsymbol{y})\boldsymbol{w} \neq \boldsymbol{s}$ be an equational problem s.t. $\boldsymbol{w}$ is a vector of free variables. Moreover, let $\boldsymbol{u} = (u_1, \ldots, u_l)$. denote the free variables occurring in $\boldsymbol{s}$ and suppose that $\boldsymbol{u} \cap \boldsymbol{w} = \emptyset$ holds.*

Then $\mathcal{P}$ is equivalent to $\mathcal{P}' \equiv (\forall \boldsymbol{y})(\forall \boldsymbol{z})(\boldsymbol{w}, \boldsymbol{u}) \neq (\boldsymbol{s}\eta, \boldsymbol{z})$, where $\boldsymbol{z} = (z_1, \ldots, z_l)$ is a vector of fresh, pairwise distinct variables and the substitution η is defined as $\eta = \{u_1 \leftarrow z_1, \ldots, u_l \leftarrow z_l\}$.

By combining the transformation steps from the Lemmas 4.1 through 4.4, we get the following theorem on the transformation into PFE-form:

Theorem 4.1. (PFE-Form). *The transformation of equational problems from $\exists^* \forall^*$-CNF into PFE-form can be done in polynomial time.*

Proof: Let $\mathcal{P} \equiv \exists \boldsymbol{z}\, \forall \boldsymbol{y}\, \mathcal{P}_1 \wedge \ldots \wedge \mathcal{P}_n$ be an equational problem, s.t. the $\mathcal{P}_i$'s are quantifier-free disjunctions of equations and disequations. Of course, the universal quantifiers may be shifted in front of the $\mathcal{P}_i$'s, thus yielding an equivalent problem $\mathcal{P}' \equiv \exists \boldsymbol{z}\, [\forall \boldsymbol{y}\, \mathcal{P}_1] \wedge \ldots \wedge [\forall \boldsymbol{y}\, \mathcal{P}_n]$. It is then possible, to apply the transformation steps from the Lemmas 4.1 through 4.4 to every conjunct $[\forall \boldsymbol{y}\, \mathcal{P}_i]$:

By Lemma 4.1, the disequations can be simplified in such a way, that the left-hand sides of the disequations consist of pairwise distinct, existentially quantified variables, which do not occur on the right-hand side of any disequation. Lemma 4.2 can be used to eliminate all parameters from the equations that do not occur in the disequations. The remaining parameters in the equations can then be eliminated by means of Lemma 4.3. When the equations contain no more parameters, the universal quantifiers can be shifted in front of the disequations. The universally quantified disequations thus produced can be brought into the desired form via Lemma 4.4. Finally, by appropriately renaming the existentially quantified variables introduced by Lemma 4.3 and shifting them to the front of the formula, the transformation into PFE-form is finished.

The correctness of these transformation steps has been proven in the Lemmas 4.1 through 4.4. As for the complexity, note that the only operations required for this transformation basically are the computation of several most general unifiers, the application of these mgu's to the appropriate terms and checking whether certain variables occur in the resulting terms or in the unifiers themselves, respectively. Hence, all of these steps can be done in polynomial time, provided that unifiers are treated as directed acyclic graphs rather than as strings. $\diamond$

The main ideas of this transformation are illustrated in the following example:

Example 4.1. Let $\mathcal{P} \equiv \exists(z_1, z_2, z_3, z_4)\, \forall(y_1, y_2, y_3, y_4)\, [\mathcal{P}_1 \wedge \mathcal{P}_2 \wedge \mathcal{P}_3]$ be an equational problem over the Herbrand universe H with signature $\Sigma(H) = \{a, f, g\}$, where the $\mathcal{P}_i$'s are defined as follows:

$$\begin{aligned}
\mathcal{P}_1 \equiv\ & f(z_1, g(z_2)) = f(y_1, z_3) \vee g(y_1) = z_3 \vee f(a, y_2) = f(z_2, g(y_4)) \vee \\
& f(g(y_2), z_1) \neq f(y_3, g(y_1)) \vee g(z_3) \neq z_2 \\
\mathcal{P}_2 \equiv\ & f(y_1, a) = f(g(z_2), a) \vee f(g(z_4), y_1) \neq f(z_2, a) \vee g(y_1) \neq g(g(y_3)) \\
\mathcal{P}_3 \equiv\ & f(g(z_2), z_1) = f(g(y_1), y_1) \vee g(y_2) = y_3 \vee f(a, y_4) = f(z_2, g(y_2)) \vee \\
& f(f(a, y_3), g(y_2)) \neq f(z_1, y_4) \vee g(z_2) \neq g(f(a, z_3)) \vee z_4 \neq y_1
\end{aligned}$$

After shifting the universal quantifiers in front of the $\mathcal{P}_i$'s, we can apply the transformation steps from the Lemmas 4.1 through 4.4 to the resulting subproblems $[\forall(y_1, y_2, y_3, y_4)\, \mathcal{P}_i]$: Note that no unifier exists for the negated disequations from $\mathcal{P}_2$, while the negated disequations of the other subproblems are unifiable with the mgu's $\vartheta_1 = \{z_1 \leftarrow g(y_1), y_3 \leftarrow g(y_2), z_2 \leftarrow g(z_3)\}$ and $\vartheta_3 = \{z_1 \leftarrow f(a, y_3), z_2 \leftarrow f(a, z_3), y_4 \leftarrow g(y_2), y_1 \leftarrow z_4\}$, respectively. Hence, by Lemma 4.1, the $\mathcal{P}_i$'s may be transformed as follows:

$$\mathcal{P}_1^{(1)} \equiv f(g(y_1), g(g(z_3))) = f(y_1, z_3) \vee g(y_1) = z_3 \vee f(a, y_2) = f(g(z_3), g(y_4))$$

$$\begin{aligned}
&\vee z_1 \neq g(y_1) \vee z_2 \neq g(z_3)\\
\mathcal{P}_2^{(1)} &\equiv \top\\
\mathcal{P}_3^{(1)} &\equiv f(g(f(a,z_3)), f(a,y_3)) = f(g(z_4), z_4) \vee g(y_2) = y_3 \vee\\
&\quad f(a, g(y_2)) = f(z_2, g(y_2)) \vee z_1 \neq f(a,y_3) \vee z_2 \neq f(a,z_3)
\end{aligned}$$

Let e_{ij} denote the j-th equation in the problem $\mathcal{P}_i$ and let ϑ_{ij} denote the mgu of e_{ij}. Then the following mgu's exist:

$$\begin{aligned}
&\vartheta_{12} = \{z_3 \leftarrow g(y_1)\} \quad && \vartheta_{31} = \{z_4 \leftarrow f(a,y_3), z_3 \leftarrow y_3\}\\
&\vartheta_{32} = \{y_3 \leftarrow g(y_2)\} && \vartheta_{33} = \{z_2 \leftarrow a\}
\end{aligned}$$

By Lemma 4.2, the equations e_{11} and e_{13} may be deleted, since they are not unifiable. Likewise, the equation e_{32} may be deleted, since it contains the parameter y_2, which does not occur in the disequations. Finally, in equation e_{33}, the parameter y_2 may be replaced by the constant symbol a. We thus get the following problems:

$$\begin{aligned}
\mathcal{P}_1^{(2)} &\equiv g(y_1) = z_3 \vee z_1 \neq g(y_1) \vee z_2 \neq g(z_3)\\
\mathcal{P}_3^{(2)} &\equiv f(g(f(a,z_3)), f(a,y_3)) = f(g(z_4), z_4) \vee f(a,g(a)) = f(z_2, g(a)) \vee\\
&\quad z_1 \neq f(a,y_3) \vee z_2 \neq f(a,z_3)
\end{aligned}$$

By Lemma 4.3, the problems $[\forall(y_1,y_2,y_3,y_4)\,\mathcal{P}_i^{(2)}]$ can be further transformed into the following problems $\mathcal{P}_i^{(3)}$ (Note that, for simplicity, we omit all variables y_j from the universal quantifier prefix that have already been eliminated from the corresponding formula. Moreover, we use term tuples as a short-hand notation for conjunctions of equations and disjunctions of disequations, respectively):

$$\begin{aligned}
\mathcal{P}_1^{(3)} &\equiv (\exists y_1)(z_1, z_2, g(y_1)) = (g(y_1), g(z_3), z_3) \vee (\forall y_1)\,(z_1,z_2) \neq (g(y_1), g(z_3))\\
\mathcal{P}_3^{(3)} &\equiv (\exists y_3)(z_1, z_2, f(g(f(a,z_3)), f(a,y_3))) = (f(a,y_3), f(a,z_3), f(g(z_4), z_4))\\
&\quad \vee (z_1, z_2, f(a,g(a))) = (f(a,y_3), f(a,z_3), f(z_2, g(a)))\\
&\quad \vee (\forall y_3)\,(z_1,z_2) \neq (f(a,y_3), f(a,z_3))
\end{aligned}$$

It is now possible to shift the universal quantifiers in front of the disequations. We can then apply Lemma 4.4 to get the following equational problems:

$$\begin{aligned}
\mathcal{P}_1^{(4)} &\equiv (\exists y_1)(z_1, z_2, g(y_1)) = (g(y_1), g(z_3), z_3) \vee\\
&\quad (\forall y_1, v)\,(z_1, z_2, z_3) \neq (g(y_1), g(v), v)\\
\mathcal{P}_3^{(4)} &\equiv (\exists y_3)(z_1, z_2, f(g(f(a,z_3)), f(a,y_3))) = (f(a,y_3), f(a,z_3), f(g(z_4), z_4))\\
&\quad \vee (z_1, z_2, f(a,g(a))) = (f(a,y_3), f(a,z_3), f(z_2, g(a)))\\
&\quad \vee (\forall y_3, v)\,(z_1, z_2, z_3) \neq (f(a,y_3), f(a,v), v)
\end{aligned}$$

After renaming the existentially quantified variables y_1 in $\mathcal{P}_1^{(4)}$ and y_3 in $\mathcal{P}_3^{(4)}$ by the fresh variables u_1 and u_2, respectively, we can shift all existential quantifiers to the front of the conjunction $\mathcal{P}_1^{(4)} \wedge \mathcal{P}_3^{(4)}$. The desired formula in PFE-form is then $\mathcal{P}' \equiv \exists(z_1, z_2, z_3, z_4, u_1, u_2)[\mathcal{P}_1^{(5)} \wedge \mathcal{P}_3^{(5)}]$ with

$$\begin{aligned}
\mathcal{P}_1^{(5)} &\equiv (z_1, z_2, g(u_1)) = (g(u_1), g(z_3), z_3) \vee\\
&\quad (\forall y_1, v)\,(z_1, z_2, z_3) \neq (g(y_1), g(v), v)\\
\mathcal{P}_3^{(5)} &\equiv (z_1, z_2, f(g(f(a,z_3)), f(a,u_2))) = (f(a,y_3), f(a,z_3), f(g(z_4), z_4))\\
&\quad \vee (z_1, z_2, f(a,g(a))) = (f(a,u_2), f(a,z_3), f(z_2, g(a)))\\
&\quad \vee (\forall y_3, v)\,(z_1, z_2, z_3) \neq (f(a,y_3), f(a,v), v)
\end{aligned}$$

5 NP-Membership

The transformation from $\exists^* \forall^*$-CNF into PFE-form from Section 4 forms the basis of our NP-membership proof for the satisfiability problem of equational problems in $\exists^* \forall^*$-CNF. In fact, by the polynomial time complexity of this transformation, we can restrict ourselves w.l.o.g. to the case of equational problems in PFE-form. In this section we shall extend the ideas developed in [7] for proving the coNP-completeness of the term tuple cover problem to equational problems in PFE-form. The key part of the coNP-membership proof in [7] is an appropriate representation of the complement of a tuple $\boldsymbol{t}$ (w.r.t. a given Herbrand universe H). In fact, the tuples $\boldsymbol{s}$ from the complement of $\boldsymbol{t}$ can be obtained as follows: Consider the tree representation of $\boldsymbol{t}$, "deviate" from this representation at some node and close all other branches of $\boldsymbol{s}$ as early as possible with new, pairwise distinct variables. The result of such a deviation is denoted as $dev_{(p,q)}(\boldsymbol{t})$: Intuitively, p indicates the position *where* one deviates from $\boldsymbol{t}$ and q tells us *how* one deviates:

- If p refers to a node labelled by some function symbol f (constants are considered as function symbols of arity 0), then $dev_{(p,q)}(\boldsymbol{t})$ is defined, iff q is a function symbol different from f with arity $\alpha(q)$. In this case, $dev_{(p,q)}(\boldsymbol{t})$ is constructed from $\boldsymbol{t}$ by replacing the subterm $[\boldsymbol{t}\,|p]$ at position p by the term $q(z_1, \ldots, z_{\alpha(q)})$ and by closing all other branches as early as possible with new variables.
- If the node (in the tree representation of $\boldsymbol{t}$) at position p is labelled by a variable y which occurs somewhere else in $\boldsymbol{t}$, then $dev_{(p,q)}(\boldsymbol{t})$ is defined, iff q is such a position in $\boldsymbol{t}$ where y also occurs. Then $dev_{(p,q)}(\boldsymbol{t})$ is constructed from $\boldsymbol{t}$ by replacing y at position p with a fresh variable z and by restricting the ground instances of the resulting term tuple through the constraint $z \neq y$. Again, all other branches are closed as early as possible with new variables.
- If the node (in the tree representation of $\boldsymbol{t}$) at position p is labelled by a variable that occurs only once in $\boldsymbol{t}$, then no deviation at all is possible at position p and, therefore, $dev_{(p,q)}(\boldsymbol{t})$ is undefined for every q.

A formal definition of this idea and the proof that the complement of a term tuple $\boldsymbol{t}$ is actually captured by these deviations at nodes of the tree representation of $\boldsymbol{t}$ can be found in [7]. This idea is illustrated in the following example:

Example 5.1. Let $\boldsymbol{t} = (f(y_1, y_2), g(y_1))$ be a term tuple over the Herbrand universe H with signature $\Sigma = \{f, g, a\}$.

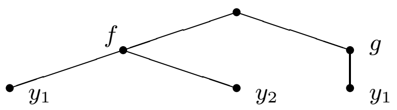


Fig. 1. tree representation of $(f(y_1, y_2), g(y_1))$

Then the tree corresponding to $\boldsymbol{t}$ is depicted in figure 1. Note that no deviation is possible at position 1.2, since $y_2 = [\boldsymbol{t}\,|1.2]$ is a variable occurring only once in $\boldsymbol{t}$. For all other positions p in $\boldsymbol{t}$, $dev_{(p,q)}(\boldsymbol{t})$ is defined for appropriately chosen q. Hence, every ground term tuple $\boldsymbol{s}$ from the complement of $\boldsymbol{t}$ is an instance of one of the following (possibly constrained) tuples:

$$\begin{array}{ll} dev_{(1,a)} = (a, v) & dev_{(2,f)} = (v, f(z_1, z_2)) \\ dev_{(1,g)} = (g(z), v) & dev_{(1.1,2.1)} = [(f(z, y_2), g(y_1)) : z \neq y_1] \\ dev_{(2,a)} = (v, a) & dev_{(2.1,1.1)} = [(f(y_1, y), g(z)) : z \neq y_1] \end{array}$$

This idea of representing the complement of a term tuple by a set of (possibly constrained) tuples can be used directly for eliminating all parameters from an equational problem in PFE-form as the following example illustrates.

Example 5.2. Let $\Sigma = \{f, g, a\}$ again denote the signature of H and let $\mathcal{P} \equiv (\forall \boldsymbol{y})(x_1, x_2) \neq \boldsymbol{t}$ denote an equational problem with $\boldsymbol{t} = (f(y_1, y_2), g(y_1))$. Then the representation of the complement of $\boldsymbol{t}$ from Example 5.1 yields the following parameter-free problem $\mathcal{P}'$, which is equivalent to $\mathcal{P}$:

$$\begin{aligned} \mathcal{P}' \equiv\ & (\exists v)(x_1, x_2) = (a, v) \vee (\exists v)(\exists z)(x_1, x_2) = (g(z), v) \vee \\ & (\exists v)(x_1, x_2) = (v, a) \vee (\exists v)(\exists z_1)(\exists z_2)(x_1, x_2)(v, f(z_1, z_2)) \vee \\ & (\exists z)(\exists y_1)(\exists y_2)[(x_1, x_2) = (f(z, y_2), g(y_1)) \wedge z \neq y_1] \vee \\ & (\exists z)(\exists y_1)(\exists y_2)[(x_1, x_2) = (f(y_1, y_2), g(z)) \wedge z \neq y_1] \end{aligned}$$

This idea of representing a universally quantified disequation by a parameter-free disjunction can be used for proving the following NP-membership result:

Lemma 5.1. (NP-Membership of PFE-Form). *The satisfiability problem for equational problems in PFE-form according to Definition 4.1 is in NP.*

Proof (Sketch): Let $\mathcal{P} \equiv (\exists \boldsymbol{x})[(e_{11} \vee \ldots \vee e_{1k_1}) \vee (\forall \boldsymbol{y}_1)(\boldsymbol{z}_1 \neq \boldsymbol{t}_1)] \wedge \ldots \wedge [(e_{n1} \vee \ldots \vee e_{nk_n}) \vee (\forall \boldsymbol{y}_n)(\boldsymbol{z}_n \neq \boldsymbol{t}_n)]$ be an equational problem s.t. the e_{ij}'s are equations, the $\boldsymbol{z}_i$'s are tuples of existentially quantified variables from $\boldsymbol{x}$ and the $\boldsymbol{t}_i$'s are term tuples with variables in $\boldsymbol{y}_i$. Then the following non-deterministic algorithm checks in polynomial time that $\mathcal{P}$ is satisfiable.

1. For every $i \in \{1, \ldots, n\}$, either guess an equation $e_{i\alpha_i}$ in the i-th disjunction with $\alpha_i \in \{1, \ldots, k_i\}$ or a (possibly constrained) tuple $dev_{(p_i,q_i)}(\boldsymbol{t}_i)$ from the complement of $\boldsymbol{t}_i$.
2. Define the conjunction $\mathcal{Q} \equiv (\exists \boldsymbol{x})\ \mathcal{Q}_1 \wedge \ldots \wedge \mathcal{Q}_n$ of equations and disequations in the following way: If in the first step an equation $e_{i\alpha_i}$ was guessed, then $\mathcal{Q}_i \equiv e_{i\alpha_i}$. If a tuple $\boldsymbol{s}_i$ without constraints from the complement of $\boldsymbol{t}_i$ was guessed, then $\mathcal{Q}_i \equiv (\exists \boldsymbol{w}_i)\boldsymbol{z}_i = \boldsymbol{s}_i$, where $\boldsymbol{w}_i$ denotes the variables in $\boldsymbol{s}_i$. Finally, if a constrained tuple $[\boldsymbol{s}_i : u \neq v]$ from the complement of $\boldsymbol{t}_i$ was guessed, then $\mathcal{Q}_i \equiv (\exists \boldsymbol{w}_i)(\boldsymbol{z}_i = \boldsymbol{s}_i \wedge u \neq v)$, where $\boldsymbol{w}_i$ again denotes the variables in $\boldsymbol{s}_i$.
3. Rename all variables in $\mathcal{Q}$ appropriately apart s.t. the existential quantifiers may be shifted in front of the conjunction.

4. Check the satisfiability of the resulting conjunction of equations and disequations via the test from Lemma 3.1, i.e.: check that the mgu ϑ of the equations exists and that the application of ϑ to the disequations does not produce a trivial disequation of the form $\boldsymbol{t} \neq \boldsymbol{t}$.

The non-deterministic guess of a disjunct in step 1 corresponds to the distributivity of $\vee$ and $\wedge$. The variable renaming and quantifier shifting in step 3 is not problematical at all and the satisfiability test for a parameter-free conjunction of equations and disequations in step 4 has already been discussed in Lemma 3.1. Hence, the only critical part for the correctness of the above algorithm is step 2. But the correctness of the representation of the complement has been proven in [7] and the correctness of the transformation of a disequation $(\forall \boldsymbol{y}_i)(\boldsymbol{z}_i \neq \boldsymbol{t}_i)$ into a parameter-free disjunction follows easily by the correspondence between the term tuple cover problem and equational problems.

The crucial point for the polynomial time complexity of the algorithm is that the size of the terms involved in the complement of a tuple $\boldsymbol{t}_i$ depends polynomially on the size of an input problem $\mathcal{P}$ even if the terms in $\mathcal{P}$ are represented as dags (= directed acyclic graphs). This is due to the fact that $dev_{(p,q)}(\boldsymbol{t}_i)$ is either defined around one path (if p is a non-variable position) or two paths (if p is a variable position) of $\boldsymbol{t}_i$, i.e.: the number of positions (which corresponds to the number of nodes in the tree representation) in the terms occurring in $dev_{(p,q)}(\boldsymbol{t}_i)$ is linearly bounded in the term depth of $\boldsymbol{t}_i$ (where the multiplicative constant is basically the maximum arity of the function symbols in H). Hence, even if the string representation of terms represented by dags may have exponential size, the size of the string representation of the (possibly constrained) tuple $dev_{(p_i,q_i)}(\boldsymbol{t}_i)$ is polynomially bounded w.r.t. the dag representation of $\boldsymbol{t}_i$. Moreover, the satisfiability test from Lemma 3.1 can be done in polynomial time independently of the chosen term representation. Therefore, the overall complexity of this non-deterministic algorithm is polynomial. $\diamond$

But then the following theorem follows immediately:

Theorem 5.1. (NP-Completeness of Equational Problems in $\exists^* \forall^*$-CNF). *The satisfiability problem for equational problems in $\exists^* \forall^*$-CNF over an infinite Herbrand universe is NP-complete.*

6 Conclusions and Future Work

The **main result** obtained in this paper is the proof of the NP-completeness (and, in particular, of the NP-membership) of the satisfiability problem for equational problems in $\exists^* \forall^*$-CNF over an infinite Herbrand universe. The key idea for this result is the polynomial time transformation of equational problems from $\exists^* \forall^*$-CNF into PFE-form according to Definition 4.1.

As far as future research in this area is concerned, **extensions and further improvements** of the satisfiability test for equational problems are the most important issue:

Our NP-membership proof can be viewed as a possible step towards a *more efficient algorithm*. Moreover, the transformation into PFE-form from Section 4 can be easily extended to a transformation into a collection of equational problems in coTTC-form according to Definition 3.2. It is thus possible to apply efficient algorithms for the well-studied term tuple cover problem to equational problems, e.g.: The algorithm from [3] as well as a deterministic version of our NP-algorithm from Theorem 5.1 has exponential complexity w.r.t. the size of the terms involved (in particular, w.r.t. the term depth). In contrast, a translation of the term tuple cover algorithm from [13] is exponential in the total number of equations and disequations, while the size of the terms only has polynomial influence on the overall complexity.

One important extension of our results concerns the study of *many-sorted universes*. It seems as though such an extension is not problematical, as long as all sorts are infinite. In particular, the transformation in Section 4 relies on the assumption that all variables are interpreted over an infinite universe (otherwise, the U_4-rule from [3] used in Lemma 4.2 as well as the satisfiability test from Lemma 3.1 would not be applicable!). However, the fact that all variables take values from the same universe, does not appear to be essential. Of course, the details of this claim have to be worked out yet.

In this paper, only equational formulae in CNF with quantifier prefix $\exists^* \forall^*$ have been investigated. This restriction is somehow justified since, in many important applications, equational formulae of this form occur in a natural way. Nevertheless, an extension of our algorithm to *arbitrary equational formulae* would be desirable. In [2], an algorithm is presented which neither requires a CNF nor a specific quantifier prefix as an input. Instead, distributivity rules are applied whenever this is necessary. Moreover, a whole collection of rules dealing with single quantifiers and combinations of quantifiers are provided. Note that equational problems in DNF can be easily shown to be Σ_p^2-hard. Equational formulae with no restriction on quantifier occurrences are even non-elementary (cf. [16]). So there is a clear limit up to which the worst case complexity can possibly be improved. However, the ideas developed in [2] may lead to significant improvements in many cases. In contrast to the extension to many-sorted universes, this kind of increased flexibility w.r.t. the form of the equational formulae seems to be much harder to integrate into our algorithm, which essentially relies on a specific form of the equational formulae under investigation. However, one may of course go in the opposite direction and integrate our transformation rules from the Lemmas 4.1 through 4.4 into the rule system from [2]. The completeness of the resulting rule system is not affected by adding new rules and the correctness of these rules has been proven in Section 4. Note that these transformations are rather cheap since they are basically made up from unification and checking for certain variable occurrences. In particular, they do not involve any exponential blow-up. Hence, adding these rules and trying to apply them before any expensive transformation rule (like the explosion rule) is applied, may possibly increase the efficiency of the algorithm from [2]. But still, the benefit from these additional rules is by no means clear and a thorough complexity analysis of the algorithm from [2] with and without our transformation rules has to be done yet.

Finally, also many *related questions concerning the complexity* of equational formulae have been left out here, e.g.: How does the NP-membership relate to the non-elementary complexity of equational formulae with arbitrary quantifier prefix (cf. [16])? What happens to the complexity, when restrictions different from the ones imposed here are considered? In particular, restricting the number of variables rather than restricting the quantifier prefix to $\exists^* \forall^*$, considering DNF rather than CNF, admitting a finite Herbrand universe, etc.

References

1. F.Baader, J.H.Siekmann: Unification Theory, in Handbook of Logic in Artificial Intelligence and Logic Programming, D.M.Gabbay, C.J.Hogger, J.A.Robinson (eds.), Oxford University Press (1994).
2. H.Comon, C.Delor: Equational Formulae with Membership Constraints, Journal of Information and Computation, Vol 112, pp. 167-216 (1994).
3. H.Comon, P. Lescanne: Equational Problems and Disunification, Journal of Symbolic Computation, Vol 7, pp. 371-425 (1989).
4. R.Caferra, N.Peltier: Extending semantic Resolution via automated Model Building: applications, Proceedings of IJCAI'95, Morgan Kaufmann, (1995)
5. R.Caferra, N.Zabel: Extending Resolution for Model Construction, in Proceedings of Logics in AI - JELIA'90, LNAI 478, pp. 153-169, Springer (1991).
6. C.Fermüller, A.Leitsch: Hyperresolution and Automated Model Building, Journal of Logic and Computation, Vol 6 No 2, pp.173-230 (1996).
7. G.Gottlob, R.Pichler: Working with ARMs: Complexity Results on Atomic Representations of Herbrand Models, to appear in Proceedings of LICS'99, IEEE Computer Society Press, (1999).
8. J.-L.Lassez, K.Marriott: Explicit Representation of Terms defined by Counter Examples, Journal of Automated Reasoning, Vol 3, pp. 301-317 (1987).
9. J.-L.Lassez, M.Maher, K.Marriott: Elimination of Negation in Term Algebras, in Proceedings of MFCS'91, LNCS 520, pp. 1-16, Springer (1991).
10. D.Lugiez: A Deduction Procedure for First Order Programs, in Proceedings of ICLP'89, pp. 585-599, Lisbon (1989).
11. M.Maher: Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees, in Proceedings of LICS'88, pp. 348-357, IEEE Computer Society Press, (1988).
12. A.Martelli, U.Montanari: An efficient unification algorithm, ACM Transactions on Programming Languages and Systems, Vol 4 No 2, pp. 258-282 (1982).
13. R.Pichler: Algorithms on Atomic Representations of Herbrand Models, in Proceedings of Logics in AI - JELIA'98, LNAI 1489, pp. 199-215, Springer (1998).
14. J.A.Robinson: A machine oriented logic based on the resolution principle, Journal of the ACM, Vol 12, No 1, pp. 23- 41 (1965).
15. T.Sato, F.Motoyoshi: A complete Top-down Interpreter for First Order Programs, in Proceedings of ILPS'91, pp. 35-53, (1991).
16. S.Vorobyov: An Improved Lower Bound for the Elementary Theories of Trees, in Proceedings of CADE-13, LNAI 1104, pp. 275-287, Springer (1996).

VSDITLU: A Verifiable Symbolic Definite Integral Table Look-Up

A. A. Adams, H. Gottliebsen, S. A. Linton, and U. Martin

Department of Computer Science, University of St Andrews
St Andrews KY16 9ES, Scotland
{aaa,hago,sal,um}@cs.st-and.ac.uk

Abstract. We present a verifiable symbolic definite integral table look-up: a system which matches a query, comprising a definite integral with parameters and side conditions, against an entry in a verifiable table and uses a call to a library of facts about the reals in the theorem prover PVS to aid in the transformation of the table entry into an answer. Our system is able to obtain correct answers in cases where standard techniques implemented in computer algebra systems fail. We present the full model of such a system as well as a description of our prototype implementation showing the efficacy of such a system: for example, the prototype is able to obtain correct answers in cases where computer algebra systems [CAS] do not. We extend upon Fateman's web-based table by including parametric limits of integration and queries with side conditions.

1 Introduction

In this paper we present a verifiable symbolic definite integral table look-up: a system which matches a query, comprising a definite integral with parameters and side conditions, against an entry in a verifiable table, and uses a call to a library of facts about the reals in the theorem prover PVS to aid in the transformation of the table entry into an answer. Our system is able to obtain correct answers in cases where standard techniques, such as those implemented in the computer algebra systems [CAS] Maple and Mathematica, do not. The importance of this work lies both in the novelty of verifiable table look up, and, more generally, as an indication of how theorem proving, particularly embedded verification with library support, can be a valuable tool to support users of mathematics, such as engineers, who want trusted results with minimal user interaction. NAG Ltd, the developers of the CAS **axiom**, brought this problem to our attention and are interested in including such a system in future projects.

Tables of mathematical formulae have been used by engineers and technicians for centuries. Inevitably such tables contained errors, sometimes slips of the pen, sometimes deliberate changes to foil copyists. In many cases accessible high-speed computation has allowed us to replace tables with on-the-fly calculation. For example a navigator's instruments and tables can now be replaced by an efficient GPS device, or an engineer's handbook with an "interactive book" based on a computer algebra system. However they are not entirely obsolete: the notorious Pentium bug was due to an error in a look-up table for SRT division, and this prompted the development by Owre and others of a general framework in the PVS prover for handling such tables [ORS97].

Definite integration, or "finding the area under a curve" is traditionally carried out using numerical techniques. However these cannot be used in the presence of parameters, and it is widely recognised in the computer algebra community [Sto91,Dav] that symbolic definite integration in the presence of parameters is a tricky problem where current algorithms are not adequate and computer algebra systems can get even very simple examples wrong. Thus table-look up is recognised as a useful solution, particularly as the answers often contain subtle side conditions. Machine look-up tables have obvious advantages over paper ones, offering automated pattern matching and simplification, ability to handle far more complex table entries and side conditions, and interoperability with other software. In particular web-based tables can be routinely updated with new results, and allow sharing and reuse of entries which may be complicated to obtain and are likely to be useful to other practitioners, as well as providing interesting opportunities for investigating user demand (which is often for nothing more difficult than homework problems). All the published paper look-up tables contain errors, so verifiable machine look up tables offer a greater possibility of freedom from error through the use of machine certification of the table entries, and their verifiable transformation to a correct answer.

In the next section we describe the problem of symbolic integration in more detail, and indicate why look-up tables are valuable. Section 3 describes the full concept for a VSDITLU, while Section 4 describes our prototype implementation, which is able to obtain correct answers in cases where standard techniques, such as those implemented in Maple and Mathematica, do not. Our prototype system extends the best available electronic table, Fateman's web-based table [EF95], by including parametric limits of integration and queries with side conditions. Section 5 addresses some of the wider issues and places our work in the context of other recent work on integrating theorem proving and computer algebra.

2 Symbolic Definite Integration

Thirty years ago an engineer wishing to compute a standard indefinite or definite integral, and preferring a trusted authority over uncertain high school math skills, would have turned to books of tables such as Gröbner and Hofreiter [GH61] or the CRC tables [ZKR96]. For example in the four hundred or so pages of [GH61] we find Entry 7 of table 1 Volume I, which expresses an indefinite integral, or more precisely an antiderivative[1], in high school math terms "a function whose

[1] Note that throughout this paper, for a a positive real, $\sqrt{a}$ denotes the positive square root of a and Log$a$ denotes the natural logarithm of a. Log denotes the principal value of the complex logarithm function.

derivative is $(ax^2 + 2bx + c)^{-1}$"

$$\int (ax^2 + 2bx + c)^{-1} \; dx = \tag{1}$$

$$\begin{array}{ll} \dfrac{\operatorname{Log}|(ax+b-p)/(ax+b+p)|}{2p} + D & \text{for } p = \sqrt{b^2 - ac}, ac < b^2 \\ \dfrac{\tan^{-1}((ax+b)/p)}{p} + D & \text{for } p = \sqrt{ac - b^2}, ac > b^2 \\ -\dfrac{1}{(ax+b)} + D & \text{for } ac = b^2 \end{array}$$

and Entry 2 of Table 13 Volume II, which expresses a definite integral, in high school math terms "the area under the curve $(ax^2 + 2bx + c)^{-1}$ for x between 0 and 1":

$$\int_0^1 (ax^2 + 2bx + c)^{-1} \; dx = \tag{2}$$

$$\begin{array}{ll} \dfrac{\operatorname{Log}|(b+c+p)/(b+c-p)|}{2p} & \text{for } p = \sqrt{b^2 - ac}, ac < b^2, c \neq 0 \\ \dfrac{\tan^{-1}((a+b)/p) - \tan^{-1}(b/p)}{p} & \text{for } p = \sqrt{ac - b^2}, ac > b^2 \\ a/b(a+b) & \text{for } ac = b^2 > 0, \text{and } b/a > 0 \text{ or } b/a < -1 \end{array}$$

To evaluate, say

$$\int_0^1 (x^2 + 2\sin(d)x + 1)^{-1} \; dx \text{ for } |d| < \pi/2 \tag{3}$$

the user matches $a = 1, b = \sin d$, $c = 1$, observes that as $1 > \sin^2 d$ the second case alone applies, simplifies under the constraint $|d| < \pi/2$ the expression in $\sin, \tan^{-1}$ that results and obtains the result $(\pi - 2d)/4\cos d$. No understanding of integration, limits, singularities of the integrand and so on is involved, just symbol manipulation to render the answer in an acceptable form.

In each case the table entry gives an indefinite or definite integral with respect to x of an integrand involving real parameters a, b and c, with limits of integration $0, 1$ in Entry 2, together with certain side conditions on each answer involving the parameters. Entry 1 is complete: the side conditions partition all possible values of a, b, c. Entry 2 is not complete: it does not cover, for example, the case $c = 0$, where in fact the integral is undefined. The entries above are correct: in 1968 Klerer and Grossman [KG68] showed that all current tables contained a small number of errors, mostly typographical: for example [ZKR96] contains a sign error in a version of Entry 1. Notice that while the left hand sides of equations such as (1), (2) are well-defined functions they may have a wide variety of representations: there is no useful notion of canonical form here and so there are in general many possible ways of expressing the table entries.

Attempts have been made to produce electronic versions of such tables, for example the CRC Standard Math Interactive CD [Zwi98], but this still contains errors, and does not offer the facilities one might expect, such as automatic matching and simplification of integrals against user input, or exporting in a standard interchange format such as OpenMath [DGW97]: not surprising when several of the formulae seem to be stored only as images! There are at least two web based look-up tables: Fateman's [FE] handles symbolic definite integrals with numeric limits of integration, includes all of the CRC entries without parametric limits of integration and is believed to be error free. The Mathematica table [Wol] is limited to indefinite integrals and calls Mathematica code which as we shall see often returns incorrect answers.

Symbolic integration algorithms inside computer algebra systems are very powerful, but as we shall indicate are currently not adequate for symbolic definite integration: hence the need for look-up tables. It is very easy for a naive user to get completely wrong answers, or get no answer at all, on input where high school techniques would find the answer fairly readily. For example even on indefinite integration Mathematica 3 returns

$$\int x^{-(a+1)/(a+1)}\ dx = \frac{x^{-(a+1)/(a+1)}}{-\frac{(a+1)}{(a+1)}+1} = x^{-1}/0$$

where the correct answer is Log$x$. It returns

$$\int (x-b)^{-1}\ dx = \mathrm{Log}(x-b),$$

without adding the side condition $x > b$ or equivalently using the more familiar answer Log$|(x-b)|$. It then evaluates the definite integral as

$$\int_0^c (x-1)^{-1}\ dx = \mathrm{Log}[c-1] - \mathrm{Log}[-1] = \mathrm{Log}[c-1] - i\pi \tag{4}$$

which gives the correct answer for $c < 1$ as the two imaginary numbers cancel out, but a complex number for $c > 1$, when the correct answer is the real number Log$|c-1|$. For simplified versions of Entry 1 Mathematica 3 returns

$$\int \frac{1}{x^2-a}\ dx = -\frac{\tanh^{-1}(x/\sqrt{a})}{\sqrt{a}}. \tag{5}$$

without side conditions: in fact while $1/(x^2-a)$ is defined except where $x^2 = a \geq 0$, the right hand side of (5) is only defined over the reals for $0 \leq x^2 < a$, where it is equal to the expression involving Log$a$ given in Entry 1. Called upon to evaluate

$$\int_{-1}^{1} \frac{1}{x^2 - \cos(a)}\ dx$$

Mathematica 3 uses Equation (5) without taking account of the possible sign change of $\cos(a)$ to get completely wrong answers. Maple V [Hec96] performs similarly. Experts can set additional flags or write further code to avoid some of these problems, but this is not straightforward for the naive user.

A full explanation for these unexpected results and how to avoid them is outwith the scope of this paper: they are consequences of implementation compromises for what is, despite the simple presentation given in high school, complex and subtle mathematics: see [Bro97,DST93]. Some would seem to be easily handled by greater care over calls to simplification routines, use of a type system such as in the computer algebra system **axiom** [JS92] or correct handling and propagation of pre- and side-conditions. Others are a consequence of problems in simplifying expressions in elementary functions[2], for which there is no canonical form or decision procedure, or in combining and simplifying parameterised expressions under constraints: for example the simplifications involved in Equation (3) defeated Maple and Mathematica.

A more fundamental problem involves the handling of functions over the reals and the blurring of computer algebra and computer analysis. CAS compute indefinite integrals by computing antiderivatives using the Risch algorithm, which involves decomposing the integrand as a sum of partial fractions. This can be expressed entirely algebraically through the theory of differential rings [Bro97]: rings with an operator satisfying $d(fg) = (df)g + f(dg)$. The Risch algorithm computes the antiderivative of a ring element f, that is an element g such that $dg = f$, generally over the complexes, where Log and $\tan^{-1}$ and so on are defined as the appropriate antiderivative: thus in this framework answers without side conditions may be correct.

There is no such framework for handling definite integrals, which are defined informally as "the area under a curve" and formally as a limit[3]. In high school we learn

$$\int_{b}^{c} f(x)\ dx = g(c) - g(b), \text{where } g \text{ is the antiderivative of } f \qquad (6)$$

and this is the formula Mathematica 3 is using in the examples above. It returns incorrect results because (6) is false in general: by the Fundamental Theorem of Calculus it is true if f is defined and continuous in $[b, c]$ and there are straightforward modifications when f is piecewise continuous in $[b, c]$ or undefined at the endpoints of the interval: of course (6) may happen to give the right answer if none of these conditions is satisfied. Thus Mathematica gets (4) wrong because

[2] Elementary functions are those built up over the reals by combinations of *log* and *exp*, and include the usual trigonometric functions and their inverses

[3] For the purposes of this paper we assume the Lebesgue definition of integral.

$1/(x-c)$ is discontinuous at $x = c$ where it has a pole (i.e. it "goes to infinity"), and (6) does not hold for $c > 1$. A correct symbolic definite integration procedure needs to work not only algebraically, as in the Risch algorithm, but analytically as well: treating poles, zeros and domains of definition of elementary functions such as Log and $\tan^{-1}$, and here computational techniques are far less well-developed. In particular since continuity is undecidable any algorithm must work with more tractable stronger pre-conditions: for example using a syntactic decomposition of the function to check for potential poles [Dup98].

This illustrates a more general design issue: there are many examples of processes, like definite symbolic integration via the Fundamental Theorem of Calculus, where a CAS may be able to compute an answer, sometimes correct, on a large class of inputs (any function where Risch returns an indefinite integral), be provably sound on only a subclass of those inputs (where the function is continuous) and be able to check soundness easily on a smaller subclass still (functions with no potential poles). Thus the suppression of pre- and side-conditions is a design decision for ease of use. Some CAS such as **axiom** are cautious: only giving an answer when pre-conditions are satisfied. Others try and propagate the side conditions to inform the user, though this can rapidly lead to voluminous output. Mathematica and Maple generally attempt to return an answer whenever they can and leave to the user the burden of checking correctness.

The general problem of implementing symbolic definite integration is hard, calls upon many issues in the foundation of analysis and algebra, and involves both calculation (for example the algebraic factorisation and simplification required in the Risch algorithm) and proof (to verify the precondition to (6)). A fully verifiable implementation would involve a fully verified implementation of the major part of a computer algebra system, as well as several graduate level textbooks such as [Bro97].

Davenport [Dav] has described a general plan for symbolic definite computation: roughly speaking this analyses the function for possible poles, uses these to decompose the range of integration into components on which the integrand is continuous, applies the Risch algorithm and the Fundamental Theorem of Calculus on each component, then combines and simplifies the answers, all in the presence of parameters. We are currently implementing this using a combination of symbolic computation and theorem proving and expect better results than would be presently possible using computer algebra techniques alone. However this approach, while it is an exciting challenge for theorem proving research, seems hard to fully automate in general, and does not really solve the problem of our putative engineer who wishes to replace paper tables with a reliable automatic machine service, usually handling different instantiations of common integral schema. Thus we are led to consider the problem of a verifiable symbolic definite integral table look-up, which validates particular table entries and hence bypasses the difficulty of verifying the integration algorithm or calculating the correct definite integral.

Indefinite integration, the fundamental theorem of calculus and so forth have been developed in several theorem provers as part of a development of theories of real analysis, for example AUTOMATH [dB80], Mizar [Try78], HOL-light [Har98] and PVS [Dut96]. However such a development does not generally enable us to calculate anything but the simplest integrals. Harrison and Théry [HT94] experimented with combining such a development with the use of a computer algebra system as an oracle to compute indefinite integrals which were then verified by the prover. However this only helps if the computer algebra system gets the integral right!

3 The VSDITLU

We describe the principle of the VSDITLU, and our prototype implementation, and discuss the theorem proving tasks it generates. These can be characterised as

- validating the table entries, which is generally an expert interactive theorem proving task
- matching a query against the table
- verifying side conditions to return a result, generally an automated theorem proving task

We are considering expressions of the form

$$\int_b^c f(x, p_1, \ldots, p_n)dx$$

where x is a real variable, $b, c, p_1, \ldots, p_n$ are real [4] parameters, and f is a function over the reals.

The VSDITLU comprises a table of validated entries of the form

$$\int_b^c f(x, p_1, \ldots, p_n)dx : K, C \tag{7}$$

where $f(x, p_1, \ldots, p_n)$ is the integrand and K is a sequence of pairs of the form $\langle A, R\rangle$. Informally such a pair denotes that, under the constraints or side conditions R,

$$\int_b^c f(x, p_1, \ldots, p_n)dx = A,$$

while C records information to assist in verifying the table entry. More precisely A is a real expression, or "unknown" or "undefined", R is a boolean combination of equalities and pure inequalities over $\{b, c, p_1, \ldots, p_n\}$ and C is a certificate, a

[4] In principle we could include additional type constraints such as integer or rational.

set of assertions for use in validating the entry. Formally the table entry asserts that for all values of the parameters b, c, p_i, and for each $\langle A, R\rangle$ in K we have

$$C \wedge (R \Longrightarrow \int_b^c f(x, p_1, \ldots, p_n)dx = A).$$

A table entry is said to be complete if it covers all possible values of the parameters, that is to say $\{\overline{R} \mid \langle A, R\rangle \in K\}$ partitions the parameter space, where $\overline{R}$ denotes the solutions of R. As we indicated above even complete table entries need not be unique. We discuss below the validation of table entries.

Figure 1 shows a typical table entry, omitting the certificates C.

$$\int_b^c \frac{1}{p+qx}\ dx = \left\{ \begin{array}{l|l} \underline{\mathit{Answer}} & \underline{\mathit{Constraints}} \\ \hline 0 & (b = c) \\ \hline \mathit{undefined} & \begin{array}{l}(q \neq 0) \wedge (b \neq c) \wedge \\ ((b = -\frac{p}{q}) \vee (c = -\frac{p}{q}))\end{array} \\ \hline \dfrac{\mathrm{Log}|qc+p| - \mathrm{Log}|qb+p|}{q} & \begin{array}{l}(q \neq 0) \wedge (b \neq c) \wedge \\ (b \neq -\frac{p}{q}) \wedge (c \neq -\frac{p}{q})\end{array} \\ \hline \dfrac{c-b}{p} & (b \neq c) \wedge (p \neq 0) \wedge (q = 0) \\ \hline \mathit{undefined} & (b \neq c) \wedge (p = 0) \wedge (q = 0) \end{array} \right.$$

Fig. 1. A Typical VSDITLU Entry

To use the table the user submits a query

$$\left(\int_{b'}^{c'} g(x, p'_1, \ldots, p'_n)\ dx, Q \right), \tag{8}$$

where $b', c', p'_1, \ldots, p'_n$ are real valued parameters and Q is a boolean combination of equalities and pure inequalities over $\{b', c', p'_1, \ldots, p'_n\}$. The integral is matched automatically against the integrand of one or more table entries of the form (7) to obtain a match, or more generally a set of matches, Θ. We discuss the matching in more detail below.

So for example, a user might enter the query:

$$\left(\int_l^m \frac{1}{cos(d) + 2x}\ dx, (m > 3) \wedge (l > 3) \right)$$

which the VSDITLU should match against the table entry in Figure 1 with $\phi = \{cos(d) \leftarrow p, 2 \leftarrow q, l \leftarrow b, m \leftarrow c\}$.

Having obtained the matchings, Θ and Q are used to return an answer

$$\int_{b'}^{c'} g(x, p'_1, \ldots, p'_n)\ dx : L$$

where L is a set of pairs of the form $\langle A', R' \rangle$, for A' a real expressions and R' a set of constraints. This denotes that for all values of the parameters b', c', p'_i, and for each $\langle A', R' \rangle$ in L we have

$$(R' \wedge Q) \Longrightarrow \int_{b'}^{c'} g(x, p'_1, \ldots, p'_n)\ dx = A'.$$

To solve for L we note first that for any $\langle A, R \rangle \in K$ and $\theta \in \Theta$ we have, for all values of the parameters b', c', p'_i, that

$$R\theta \Longrightarrow \int_{b'}^{c'} g(x, p'_1, \ldots, p'_n)\ dx = A\theta,$$

and hence

$$R\theta \wedge Q \Longrightarrow \int_{b'}^{c'} g(x, p'_1, \ldots, p'_n)\ dx = A\theta \tag{9}$$

and so if K is complete we may take

$$L = \{\langle A\theta, R\theta \wedge Q \rangle \mid \langle A, R \rangle \in K, \theta \in \Theta\}.$$

However if for some θ and R the set of constraints $R\theta \wedge Q$ has no solutions in b', c', p'_i, that is if

$$\neg \exists b', c', p'_i \ .\ R\theta \wedge Q, \tag{10}$$

then $\langle A\theta, R\theta \wedge Q \rangle$ contributes no extra solutions to L, and so we may assume

$$L = \{\langle A\theta, R\theta \wedge Q \rangle \mid \langle A, R \rangle \in K, \theta \in \Theta, (\exists b', c', p'_i \ .\ R\theta \wedge Q)\}.$$

Thus each $\langle A, R \rangle \in K$ gives rise to a possible side condition (10), and if this side condition can be proved $\langle A, R \rangle$ does not contribute to L. If the side condition cannot be proved even though the assertion is true, then $\langle A\theta, R\theta \wedge Q \rangle$ remains in L but is redundant.

In our example we have five such side conditions:

$$\begin{aligned}
&\neg \exists l, m, d.[b = c]\phi \wedge l > 3 \wedge m > 3,\\
&\neg \exists l, m, d.[(q \neq 0) \wedge (b \neq c) \wedge ((b = -\frac{p}{q}) \vee (c = -\frac{p}{q}))]\phi \wedge l > 3 \wedge m > 3,\\
&\neg \exists l, m, d.[(q \neq 0) \wedge (b \neq c) \wedge (b \neq -\frac{p}{q}) \wedge (c \neq -\frac{p}{q})]\phi \wedge l > 3 \wedge m > 3,\\
&\neg \exists l, m, d.[(b \neq c) \wedge (p \neq 0) \wedge (q = 0)]\phi \wedge l > 3 \wedge m > 3,\\
&\neg \exists l, m, d.[(b \neq c) \wedge (p = 0) \wedge (q = 0)]\phi \wedge l > 3 \wedge m > 3.
\end{aligned}$$

Of these the second, fourth and fifth are true, essentially as $-1 \leq \cos(d) \leq 1$, and so we obtain the answer

$$\int_{l}^{m} \frac{1}{cos(d)+2x}\,dx = \begin{cases} \underline{\mathit{Answer}} & \underline{\mathit{Constraints}} \\ \hline 0 & (l = m > 3) \\ \hline \mathrm{Log}\dfrac{|2m + cos(d)|}{|2l + cos(d)|} & (l \neq m) \wedge (l > 3) \wedge (m > 3) \end{cases}$$

Notice that for concision we have simplified the constraints $R\phi \wedge Q$ remaining in the answer. As before there is no canonical way to do this, though there are sometimes obvious redundancies and subsumptions to be eliminated.

We now discuss the theorem proving tasks in more detail. Rather than work with a precise class of functions (for example linear or polynomial), when we could develop a clear theoretical analysis of the scope and limitations of our methods, we have deliberately put no restrictions on the real functions that can occur in the table entries. Matters are undecidable in general, and we necessarily can only give a rather vague account of the scope of our techniques, trusting rather on implementing a range of methods which can cover tractably the sort of input that is likely to occur in practice (not too deeply nested for example).

Validating the Table Entries

To add a new entry to the table it is necessary to provide K and to verify that the entry is correct. There are two parts to this verification:

– showing that the entry (7) is correct
– showing that the entry is complete

In the previous section we described what would be involved in computing and verifying an integral from scratch via the Risch algorithm. What we propose here is more straightforward: we assume that the computation has already been done, possibly by ad hoc means or by calling upon an existing table, and all that is required is to validate the result. The certificate C allows us to provide auxiliary lemmas to assist in this. Thus for example the part of the certificate covering the second line of (2) is just the assertion that $(ax^2 + 2bx + c)^{-1}$ is continuous in $[0, 1]$. To verify this part of the entry we need first to verify the indefinite integral of Entry (1) (which we can do by differentiating it), then to verify the certificate, and then we may invoke the Fundamental Theorem of Calculus (6) to verify the answer. Nonetheless it is still unlikely that except in the very simplest of cases this verification could be carried out automatically unless each certificate included a proof outline drawn up by a domain expert: the proof needs to call on a rich lemma database of facts about continuity, singularities, elementary functions and so forth.

We have not yet worked out a format for C suitable for a production version of the VSDITLU. It seems likely that proof planning [KKS96] will be useful here: the certificate might comprise a full proof plan or a standard template with information about poles and so forth for use with a pre-prepared proof plan.

The second part of this verification involves showing completeness by showing that $\{\overline{R} \,|\, \langle A, R\rangle \in K\}$ partitions the parameter space, where $\overline{R}$ denotes the set of solutions of R. If the number of cases is small this is a straightforward task for PVS, particularly if the constraints are linear and fall to its built in linear arithmetic package. We may reduce the theorem proving task by requiring only that $\{\overline{R} \,|\, \langle A, R\rangle \in K\}$ covers the parameter space, in which case we may have redundancy in our table entry, but it will still cover all cases. We have not yet addressed the problem of partial subsumption or overlap between different entries in the table: this comes back again to problems of representation.

Matching. The most general form of matching here is undecidable: we are working over the reals and so for example $x+2$ needs to match $b+x+3$, $x-1/b$ $x+b^2$ and $x+1/b^2$ but not $x-b^2$ or $x-1/b^2$. The best we can hope for is a suite of methods sufficient to cover a wide range of cases: it is common also in computer algebra systems to make the problem more tractable by pre-processing functions to a standardised form, for example $x+a^2$ is represented as $x+c$ with the side condition $a^2 = c$. In addition certain forms, such as sums, tend not to occur in integral tables as it is assumed the user has pre-processed a query such as $\int (f(x) + g(x))dx$ into separate queries $\int f(x)dx$, $\int g(x)dx$. Note also that, while a query may match several entries, it is sufficient for our purposes to find a match against one complete table entry to get the required answer.

While there is a rich literature on matching and unification, as far as we know there is no existing implementation that is entirely adequate for our purposes. In his look-up table Fateman [EF95] uses pre-processing and stores the integrands in a particular kind of discrimination tree: matching is performed by a succession of approximate matches. Dalmas [DGH96], in his work on a deductive database of mathematical formulae uses a similar data-structure together with a conditional AC-unification algorithm implemented in ML (using logic programming techniques). Both systems appear to be correct but not complete, that is there are matches which they fail to find.

Proving the Side Conditions. The side conditions have the form

$$\neg\exists b, c, p_i \,.\, H$$

where H is a boolean combination of equalities and strict inequalities involving real functions over $\{b, c, p_1, \dots, p_n\}$.

For polynomial functions the problem may be addressed using quantifier elimination algorithms [Bro98] which solve the complementary problem

$$\exists b, c, p_i \,.\, H.$$

Extending these to other functions such as *exp* or *log* is currently an active research area, and in any case these methods are intractable for all but the smallest examples.

Thus we turn to theorem provers for reasoning about the reals. This can involve the development of substantial theories as found in textbooks, as was done in AUTOMATH [dB80] and Mizar [Try78]. However in practical applications such as this what is often needed is a library of more low level lemmas unlikely to be found explicitly in text books, such as $\forall x \,.\, 0 \leq cos^2(x) \leq 1$, together with a tactic mechanism which allows them to be applied automatically.

Harrison [Har98] developed a large portion of real analysis in HOL-light, both major theorems and also setting in place the mechanisms (power-series and so forth) to prove low-level lemmas about elementary functions. The reals are constructed by means of Dedekind cuts. By contrast Dutertre [Dut96] uses an axiomatic approach, extending the built in axiomatisation of the reals, to prove results about the reals in PVS, and again proves both major theorems and more low-level results. Fleuriot [FP98] has also implemented both classical and non-standard reals in Isabelle.

4 Our Implementation

Our implementation consists of a front end comprising the table entries and a matching algorithm: around 2000 lines of Allegro Common Lisp. At present table entries and calls must be in a fairly strict standard form and we do not do any additional simplifications or redundancy checks: in principle our front end could be interfaced to a computer algebra system such as **axiom** for pre- and post-processing so as to handle a wider variety of inputs. The standard form aids the matching which is currently a basic form of AC pattern matching which makes no attempt to account for the units $(1, 0)$ of the AC operators $(+, *)$. For a full description see [AGLM99]. Some of the table entries have been validated though we have not yet developed the notion of certificate very precisely.

PVS is called through emacs to prove the side conditions and return an answer. This is intended to be fully automatic: it uses a large lemma database of elementary facts about the reals, which we have built on top of Dutertre's implementation [Dut96], and uses the PVS "grind" command which applies a brute force search in an attempt to prove the required results. Initially we added properties of elementary functions as additional axioms on an ad hoc basis, but after experimenting with this we decided to re-implement Harrison's work in PVS to give a basis for proving whatever lemmas we need about elementary functions. Grind in turn is calling built-in PVS procedures to handle Boolean combinations and inequalities: if for example the inequalities happen to be linear a further efficient decision procedure can be called.

Development time was very short: 3-4 person months. Our choice of PVS was a fairly pragmatic one, based on what system had the best real library available in August 1998: more recent work in HOL makes it also a suitable candidate. Our table currently contains six entries and we have been able to evaluate correctly around 60 examples from a test suite of symbolic definite integrals that CAS running in a naive mode were unable to evaluate or got wrong.Our implementation got no answers wrong, but it did sometimes fail to return an answer because our matching algorithm was not powerful enough. On one occasion *grind* sent PVS into an apparently infinite loop and it was unable to identify an obvious counter-example to the non-existence of values for the parameters in a particular theorem.

Some of the integrals in the table are:

$$\int_b^c \frac{p}{x^2+a}\,dx\;(*)\;\int_b^c \frac{1}{x^2+a}\,dx\;(*)\;\int_a^b x\tan^{-1}(\frac{1}{x})\,dx$$

The integrals marked $(*)$ have a complete set of answers. No CAS that we tested (Maple V, Mathematica 3, **axiom** and Matlab) was able to consistently produce full correct answers to these integrals: in fact, all these CAS produce incorrect answers to some of them.

5 Discussion

The integration of computer algebra and theorem proving has attracted much research interest recently, in the form of verifying computer algebra algorithms in theorem provers [Thé98], adding inference mechanisms to CAS [CZ94], using proof planning techniques to aid in organising calculations [KKS96] or arranging for provers to make calls to CAS [HT94,HC94], either as oracles for results that are then verified or as trusted components for routine manipulations. We have argued elsewhere [Mar98] that while such endeavours are valuable as contributions to theorem proving research the resulting systems are not necessarily widely used by mathematicians as they fail to address issues of mathematical practice: how computational mathematics is actually done.

We have used automated reasoning tools to solve a problem identified by the computer algebra community, and which current computer algebra systems or look-up tables did not solve satisfactorily. Our system is designed for users of mathematics such as engineers, rather than expert mathematicians: such users want a black box system which solves the problem at hand rather than grappling with anything that requires user interaction with a theorem prover.

The success of this work leads us to suggest a way forward for the integration of computer algebra systems and theorem provers: theorem proving technology could be used to provide a variety of black box components for incorporation into applications like VSDITLU. Of particular interest would be components that enhanced, rather than duplicating, the capabilities of computer algebra systems. Theorem proving over the reals is a particularly important area as functions over floating point systems or the reals occur in a variety of engineering and safety critical applications: for example Dutertre's work was originally motivated by an avionics application. We have been using PVS as such a black box for determining if Boolean combinations of inequations and equations are solvable: other examples might include matching or unification engines for real functions, which would replace the somewhat ad-hoc matching algorithm that we implemented, engines for determining continuity of a function in a given region or engines for simplifying expressions in elementary functions.

Acknowledgements

We acknowledge support of the UK EPRSC under grant number GR/L48256 and of NAG Ltd, and we thank Mike Dewar from NAG for his interest and suggestions, James Davenport and Richard Fateman for advice on computer algebra, and Bruno Dutertre for allowing us to extend his code for the reals in PVS.

References

AGLM99. A. A. Adams, H. Gottliebsen, S. A. Linton, and U. Martin. A Verifiable Symbolic Definite Integral Table Look-Up. Technical Report CS/99/3, University of St Andrews, 1999.

Bri97. E. Brinksma, editor. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '97)*. Springer-Verlag LNCS 1217, 1997.

Bro97. M. Bronstein. *Symbolic integration I*. Springer-Verlag, Berlin, 1997. Transcendental functions.

Bro98. C. W. Brown. Simplification of truth-invariant cylindrical algebraic decompositions. In Gloor [Glo98], pages 295–301.

Bun94. A. Bundy, editor. *CADE-12: 12th International Conference on Automated Deduction: Proceedings*. Springer-Verlag LNAI 814, 1994.

CC94. J. Calmet and J. A. Campbell, editors. *Integrating Symbolic Mathematical Computation and Artificial Intelligence*. Springer-Verlag LNCS 958, 1994.

CL96. J. Calmet and C. Limongelli, editors. *Design and Implementation of Symbolic Computation Systems, International Symposium, DISCO'96*. Springer-Verlag LNCS 1128, 1996.

CZ94. E. Clarke and X. Zhao. Combining symbolic computation and theorem proving: Some problems of Ramanujan. In Bundy [Bun94], pages 758–763.

Dav. J. H. Davenport. Really Strong Integration Algorithms. In preparation.

dB80. N. G. de Bruijn. A Survey of the Project AUTOMATH. In Seldin and Hindley [SH80], pages 579–606.

DGH96. S. Dalmas, M. Gaëtano, and C. Huchet. A Deductive Database for Mathematical Formulas. In Calmet and Limongelli [CL96].

DGW97. S. Dalmas, M. Gaëtano, and S. Watt. An OpenMath 1.0 Implementation. In Küchlin [Küc97], pages 241–248.

DST93. J. H. Davenport, Y. Siret, and E. Tournier. *Computer algebra*. Academic Press Ltd, London, second edition, 1993.

Dup98. B. Dupée. Using Computer Algebra to Find Singularities of Elementary Real Functions. Available from the author, bjd@maths.bath.ac.uk, 1998.

Dut96. B. Dutertre. Elements of Mathematical Analysis in PVS. In von Wright et al. [vWGH96].

EF95. T. Einwohner and R. J. Fateman. Searching techniques for Integral Tables. In Levelt [Lev95], pages 133–139.

FE. R. J. Fateman and T. Einwohner. TILU Table of Integrals Look Up. Web Service. `http://http.cs.berkeley.edu/~fateman/htest.html`.

FP98. J. Fleuriot and L. Paulson. A combination of nonstandard analysis and geometry theorem proving with application to Newton's Principia. In Kirchner and Kirchner [KK98], pages 3–16.

GH61. W. Groëbner and N. Hofreiter. *Integraltafel*. Springer-Verlag, Vienna, 1961.

Glo98. O. Gloor, editor. *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*. ACM Press, 1998.

Har98. J. Harrison. *Theorem Proving with the Real Numbers*. Springer-Verlag, 1998.

HC94. K. Homann and J. Calmet. Combining theorem proving and symbolic mathematical computing. In Calmet and Campbell [CC94], pages 18–29.

Hec96. A. Heck. *Introduction to Maple*. Springer-Verlag, New York, second edition, 1996.

HT94. J. Harrison and L. Théry. Extending the HOL theorem prover with a computer algebra system to reason about the reals. In Joyce and Seger [JS94], pages 174–185.

JS92. R. D. Jenks and R. S. Sutor. *AXIOM*. Springer-Verlag, 1992.

JS94. J. J. Joyce and C-J. H. Seger, editors. *Higher order logic theorem proving and its applications*, Berlin, 1994. Springer-Verlag LNCS 780.

KG68. M. Klerer and F. Grossman. Error Rates in Tables of Indefinite Integrals. *Journal of the Industrial Mathematics Society*, 18:31–62, 1968.

KK98. C. Kirchner and H. Kirchner, editors. *CADE-15: 15th International Conference on Automated Deduction: Proceedings*. Springer-Verlag LNAI 1421, 1998.

KKS96. M. Kerber, M. Kohlhase, and V. Sorge. Integrating Computer Algebra with Proof Planning. In Calmet and Limongelli [CL96].

Küc97. W. W. Küchlin, editor. *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*. ACM Press, 1997.

Lev95. A. H. M. Levelt, editor. *Proceedings of the 6th International Symposium on Symbolic and Algebraic Computation, ISSAC '95*. Springer-Verlag LNCS 1004, 1995.

Mar98. U. Martin. Computers, Reasoning and Mathematical Practice. In Computational Logic, NATO Adv. Sci. Inst. Ser. F Comput. Systems Sci., 1998.

ORS97. S. Owre, J. Rushby, and N. Shankar. Integration in PVS: Tables, Types, and Model Checking. In Brinksma [Bri97], pages 366–383.

SH80. J. P. Seldin and J. R. Hindley, editors. *To H.B. Curry: essays on combinatory logic, lambda calculus and formalism*. Academic Press, 1980.

Sto91. D. Stoutemyer. Crimes and misdemeanours in the computer algebra trade. *Notices of the AMS*, 38:779–785, 1991.

Thé98. L. Théry. A Certified Version of Buchberger's Algorithm. In Kirchner and Kirchner [KK98], pages 349–364.

Try78. A. Trybulec. The Mizar-QC 6000 logic information language. *ALCC Bulletin*, 6:136–140, 1978.

vWGH96. J. von Wright, J. Grundy, and J. Harrison, editors. *Theorem Proving in Higher Order Logics: 9th International Conference*. Springer-Verlag LNCS 1125, 1996.

Wol. Wolfram Research Inc. The integrator: the power to do integrals as the world has never seen before, `http://www.integrals.com`.

ZKR96. D. Zwillinger, S. G. Krantz, and K. H. Rosen, editors. *CRC standard mathematical tables and formulae*. CRC Press, Boca Raton, FL, 30th edition, 1996.

Zwi98. D. Zwillinger. Standard Math Interactive. CD-ROM, 1998.

A Framework for the Flexible Integration of a Class of Decision Procedures into Theorem Provers[⋆]

Predrag Janičić[1], Alan Bundy[2], and Ian Green[2]

[1] Faculty of Mathematics, University of Belgrade
Studentski trg 16, 11 000 Belgrade, Yugoslavia
janicic@matf.bg.ac.yu

[2] Division of Informatics, University of Edinburgh
Edinburgh EH1 1HN, Scotland
{A.Bundy,I.Green}@ed.ac.uk

Abstract. The role of decision procedures is often essential in theorem proving. Decision procedures can reduce the search space of heuristic components of a prover and increase its abilities. However, in some applications only a small number of conjectures fall within the scope of the available decision procedures. Some of these conjectures could in an informal sense fall 'just outside' that scope. In these situations a problem arises because lemmas have to be invoked or the decision procedure has to communicate with the heuristic component of a theorem prover. This problem is also related to the general problem of how to flexibly integrate decision procedures into heuristic theorem provers. In this paper we address such problems and describe a framework for the flexible integration of decision procedures into other proof methods. The proposed framework can be used in different theorem provers, for different theories and for different decision procedures. New decision procedures can be simply 'plugged-in' to the system. As an illustration, we describe an instantiation of this framework within the Clam *proof-planning* system, to which it is well suited. We report on some results using this implementation.

1 Introduction

Decision procedures have a very important role in heuristic theorem provers — they can reduce the search space of heuristic components of a prover and increase its abilities. Decision procedure can both close a branch in a proof and reject non-theorems. Some decision procedures are (generally) inefficient, but even they could increase abilities of a prover. Some decision procedures, such as a decision procedure for Presburger arithmetic [18], can have a useful role in proving some typical conjectures occurring in software and hardware verification. In some applications, decision procedures themselves cannot significantly improve the overall efficiency of a prover; for example, when only a small number of conjectures fall within the domain of the available decision procedures.

[⋆] The second and the third author are supported in part by EPSRC grant GR/M45030.

However, some (or many) of these conjectures could be in a sense "close" to that domain (see [3]). For instance, the formula

$$\forall l\, \forall \alpha\, \forall k \quad (l \leq min(\alpha) \ \wedge\ 0 < k \ \ \rightarrow \ \ l < max(\alpha) + k)$$

is not a Presburger arithmetic formula. Besides, the formula $\forall max \forall min \forall l \forall k\ (l \leq min \wedge\ 0 < k \ \rightarrow\ l < max + k)$ obtained by generalising $min(\alpha)$ to min and $max(\alpha)$ to max is a Presburger arithmetic formula, but is not a theorem. The power of decision procedures can be increased by linking them to heuristic components of the prover so they can communicate; by using already proved lemmas and by combining different decision procedures. For instance, in the given example, if the lemma $\forall \xi\ (min(\xi) \leq max(\xi))$ is available, it can be used (with the right instantiation) and lead to $\forall l \forall \alpha \forall k\ (min(\alpha) \leq max(\alpha) \rightarrow (l \leq min(\alpha) \wedge\ 0 < k\ \rightarrow\ l < max(\alpha) + k))$. After generalisation, we get the formula $\forall max \forall min \forall l \forall k\ (min \leq max \rightarrow (l \leq min \wedge\ 0 < k\ \rightarrow\ l < max + k))$ which can be proved by the decision procedure for the Presburger arithmetic. Procedures dealing with such problems are built into most state-of-the-art theorem proving systems. In [3] there is a description of one such system — a procedure for linear arithmetic based on Hodes' algorithm. However, in that paper, implementation details are mixed up with a description of the underlying algorithm. Moreover, the system itself depends to a high degree on certain specific data structures and can hardly be described without these devices. All this makes the system from [3] inapplicable in theories other than linear arithmetic. Besides, even for linear arithmetic, it is not obvious how a procedure other than that due to Hodes' could be integrated into the prover.

In this paper we give a kind of rational reconstruction of some of the approaches presented in [3]. We present a general method that can incorporate different decision procedures into a heuristic theorem prover. This general method is flexible in its structure and can be used for different decision procedures, for different theories and in different theorem provers. Within this modular system, the new decision procedure (with only a syntactical description of a corresponding theory) can be 'plugged-in' into a prover without requiring any theoretical analysis of the corresponding theory. This method is rather general and domain specific knowledge is encapsulated in smaller subprocedures specialised for certain theories.

Overview of the Paper. In section 2, we give some notation and background; we introduce the notion of a *heaviest non-$\mathcal{T}$-term* in section 3; in section 4 we give a description of modules that make up the extended proof method; in section 5 we define the proposed general method and in section 6 we consider one example from [3] in order to illustrate how the proposed method works. Section 7 discusses the termination, the soundness and some other properties of the proposed method and in section 8 we give some results of the preliminary implementation of the method. Section 9 discusses some possible refinements. In section 10 we discuss related work and in section 11 we draw some final conclusions.

2 Background and Notation

Let us define a theory $\mathcal{T}$ within first order logic with equality. Let T be some fixed set of types $\{\tau_i | i \in I\}$. For each type τ_i let V_i be a denumerable set of variables of that type ($V_i \cap V_j = \emptyset$ for $i \neq j$) and let $V = \cup_{i \in I} V_i$. Let Σ be a finite set of function symbols, each having either a type τ, $\tau \in T$ (then we call it a *constant* of type τ) or a type of the form $\tau_{i_1} \times \tau_{i_2} \times \ldots \times \tau_{i_k} \rightarrow \tau$ ($\tau_{i_1}, \ldots, \tau_{i_k}, \tau \in T$). Let Π be a finite set of predicate symbols, each having a type `truth` (with members `true` and `false`) or a type of the form $\tau_{i_1} \times \tau_{i_2} \times \ldots \times \tau_{i_k} \rightarrow \texttt{truth}$ ($\tau_{i_1}, \ldots, \tau_{i_k} \in T$). We define the notion of a $\mathcal{T}$*-term* in the following way: function symbols and variables of type τ are $\mathcal{T}$-terms of type τ; if $t_1, t_2, \ldots, t_n$ are $\mathcal{T}$-terms of types $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_k}$ respectively and a function symbol $f \in \Sigma$ has a type $\tau_{i_1} \times \tau_{i_2} \times \ldots \times \tau_{i_k} \rightarrow \tau$, then $f(t_1, t_2, \ldots, t_n)$ is a $\mathcal{T}$-term of type τ; all terms could be obtained using the first two rules. For a term which is a variable x, the set of free variables is $\{x\}$; for a term $f \in \Sigma$ the set of free variables is empty; for $f(t_1, t_2, \ldots, t_n)$ the set of free variables is the union of free variables in $t_1, t_2, \ldots, t_n$. A $\mathcal{T}$*-atomic formula* is either an expression $p(t_1, t_2, \ldots, t_n)$ (where $t_1, t_2, \ldots, t_n$ are $\mathcal{T}$-terms of types $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_k}$ respectively and $p \in \Pi$ is a predicate symbol of type $\tau_{i_1} \times \tau_{i_2} \times \ldots \times \tau_{i_k} \rightarrow \texttt{truth}$) or an expression $t_1 =_\tau t_2$ (where t_1 and t_2 are $\mathcal{T}$-terms of type τ) or a constant of type `truth`. For $p(t_1, t_2, \ldots, t_n)$ the set of free variables is the union of free variables in $t_1, t_2, \ldots, t_n$; for $t_1 =_\tau t_2$ the set of free variables is the union of free variables in t_1 and t_2. If F_1 is a $\mathcal{T}$-atomic formula, then it is also a $\mathcal{T}$*-formula.* If F_1 and F_2 are $\mathcal{T}$-formulae and $x \in V$, then also $\forall x\ F_1$, $\exists x\ F_1$, $\neg F_1$, $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, $(F_1 \rightarrow F_2)$ are $\mathcal{T}$-formulae. The set of free variables in $\forall x\ F_1$ and $\exists x\ F_1$ is the set of free variables in F_1 minus $\{x\}$. The set of free variables in $\neg F_1$ is the set of free variables in F_1. The set of free variables in $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, $(F_1 \rightarrow F_2)$ is the union of sets of free variables in F_1 and F_2. A $\mathcal{T}$-formula with no free variables we call a *closed* $\mathcal{T}$*-formula* or a $\mathcal{T}$*-sentence.* Further, by $\mathcal{T}$-formulae we will mean closed $\mathcal{T}$-formulae (unless stated otherwise). A formula F is *ground* if it has no variables. A formula F is in *prenex normal form* if it is of the form $Q_1 x_1\ Q_2 x_2\ \ldots Q_k x_k\ F'$ where $Q_i \in \{\forall, \exists\}$, $x_i \in V$ and there are no quantifiers in F'. A formula F is *universally closed* if it is of the form $\forall x_1\ \forall x_2\ \ldots \forall x_k\ F'$ where $x_i \in V$ and there are no quantifiers in F'. A formula appearing in a formula F has a *polarity* that is either positive (+) or negative (−). The top-level formula F being proved has positive polarity, written F^+. The complement of a polarity p is written $\overline{p}$, defined to be $\overline{+} = -$ and $\overline{-} = +$. Polarity is defined recursively over the structure of formulae: $(\neg F^{\overline{p}})^p$ (if p is the polarity of $\neg F$ then $\overline{p}$ is the polarity of F), $(\forall x\ F^p)^p$, $(\exists x\ F^p)^p$, $(F_1^p \wedge F_2^p)^p$, $(F_1^p \vee F_2^p)^p$, $(F_1^{\overline{p}} \rightarrow F_2^p)^p$.

A *theory* $\mathcal{T}$ (or an *axiom system* $\mathcal{T}$) is some fixed set of $\mathcal{T}$-formulae. If a formula F can be derived from that set using usual classical logic inference system we denote that by $\mathcal{T} \vdash F$ and we call the formula F a $\mathcal{T}$*-theorem* (and we say that F is *valid* in $\mathcal{T}$). Otherwise, we write $\mathcal{T} \nvdash F$ and we say that F is *invalid* in $\mathcal{T}$.

Let Σ^e, Π^e and V^e be some finite sets of function, predicate and variable symbols over some set of types T^e such that $\Sigma \subseteq \Sigma^e$, $\Pi \subseteq \Pi^e$, $V \subseteq V^e$ and

$T \subseteq T^e$. We define the notions of $\mathcal{T}^e$-term, $\mathcal{T}^e$-atomic formula, $\mathcal{T}^e$-formula and a theory $\mathcal{T}^e$ by analogy. If all formulae of the theory $\mathcal{T}$ belong to the theory $\mathcal{T}^e$, we say that $\mathcal{T}^e$ is an extension of the theory $\mathcal{T}$ (or the theory $\mathcal{T}$ is a subtheory of the theory $\mathcal{T}^e$).[1]

A theory $\mathcal{T}$ is decidable if there is an algorithm (which we call a *decision procedure*) such that for an input $\mathcal{T}$-formula F, it returns `true` if and only if $\mathcal{T} \vdash F$ (and returns `false` otherwise). For a number of decidable theories there are decision procedures that work using the idea of successive elimination of quantifiers from formula being proved ([13]). When all quantifiers are eliminated, the formulae is ground and can be easily reduced to `true` or `false`. In this paper, we will be concerned with this kind of theory.

One of such theories is Presburger Natural Arithmetic (PNA). In this theory, all variables are of type `pnat` (from *Peano NATurals*), $\Sigma = \{0, s, +\}$, $(0 : \texttt{pnat}$, $s : \texttt{pnat} \rightarrow \texttt{pnat}$, $+ : \texttt{pnat} \times \texttt{pnat} \rightarrow \texttt{pnat})$, $\Pi = \{<, >, \leq, \geq\}$ (all the predicate symbols are of the type $\texttt{pnat} \times \texttt{pnat} \rightarrow \texttt{truth}$). We write 1 instead of $s(0)$, etc. Multiplication of a variable by a constant can also be considered as in PNA: nx is treated as $x + \cdots + x$, where x appears n times. The axioms of PNA are those of Peano arithmetic without axioms concerning multiplication.

Similarly, we introduce theories *Presburger Integer Arithmetic* (PIA) and *Presburger Real Arithmetic* (PRA).[2] It was Presburger who first showed that PIA is decidable [18]. The decidability of PNA can be proved in an analogous way. PRA is also decidable [13]. The arithmetic with multiplication only is also decidable. The whole of arithmetic is known to be undecidable.

3 Heaviest Non-$\mathcal{T}$-Term

A conjecture attempted to be proved by some specific decision procedure for some theory $\mathcal{T}$ would often "fall just outside its domain", i.e., a formula F being proved could be a $\mathcal{T}^e$-formula, but not a $\mathcal{T}$-formula. In that case and if $\Pi^e \setminus \Pi$ is empty, it would mean that F involves *non-$\mathcal{T}$-terms*. For now on, we will assume $\Pi^e \setminus \Pi = \emptyset$. Recall that all function symbols from Σ^e have either a type τ or a type of the form $\tau_{i_1} \times \tau_{i_2} \times \cdots \times \tau_{i_k} \rightarrow \tau$, where $\tau \in T$, so the formula F' obtained in such a manner is a $\mathcal{T}$-formula.

Definition 1. *For $\mathcal{T}^e$-formulae and $\mathcal{T}^e$-atomic formulae, we define sets of* non-$\mathcal{T}$-terms *in the following way:*

- *the set of non-$\mathcal{T}$-terms in $\forall x\ F$, $\exists x\ F$, $\neg F$ is equal to the set of non-$\mathcal{T}$-terms in F;*

[1] Note that it can be determined whether a well-formed formula is a $\mathcal{T}$-formula or a $\mathcal{T}^e$-formula in linear time on the size of a formula.

[2] For identical and related theories a number of different terms are used. For instance, Hodes calls Presburger rational arithmetic a *theory EAR* — "the elementary theory of addition on the reals" [9]. Boyer and Moore [3] describe a universally quantified fragment of Presburger rational arithmetic as *linear arithmetic* (although in fact they work over the integers); that same theory sometimes goes by the name *Bledsoe real arithmetic.*

- *the set of non-$\mathcal{T}$-terms in* $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, $(F_1 \rightarrow F_2)$ *is equal to the union of sets of non-$\mathcal{T}$-terms in* F_1 *and* F_2*;*
- *in* $p(t_1, t_2, \ldots, t_n)$*, where* $p \in \Pi$*, the set of non-$\mathcal{T}$-terms is the union of sets of non-$\mathcal{T}$-terms in terms* t_j*,* $(j = 1, 2, \ldots, n)$*;*
- *in* $t_1 =_\tau t_2$*, the set of non-$\mathcal{T}$-terms is the set of non-$\mathcal{T}$-terms is the union of sets of non-$\mathcal{T}$-terms in terms* t_j*,* $(j = 1, 2)$*;*
- *in* `true` *and* `false` *the set of non-$\mathcal{T}$-terms is empty.*

Definition 2. *For $\mathcal{T}^e$-terms, we define sets of* non-$\mathcal{T}$-terms *in the following way:*

- *if* t *is a $\mathcal{T}$-term, then its set of non-$\mathcal{T}$-terms is empty;*
- *in* $f(t_1, t_2, \ldots, t_n)$*, where* $f \notin \Sigma$ *(and* $n \geq 0$*), the set of non-$\mathcal{T}$-terms is* $\{f(t_1, t_2, \ldots, t_n)\}$*;*
- *in* $f(t_1, t_2, \ldots, t_n)$*, where* $f \in \Sigma$ *(and* $n \geq 0$*), the set of non-$\mathcal{T}$-terms is the union of sets of non-$\mathcal{T}$-terms in terms* t_j $(j = 1, 2, \ldots, n)$*.*

In deciding whether a $\mathcal{T}^e$-formula F is a theorem, our motivation is to use a decision procedure for $\mathcal{T}$ (either by using the decision procedure itself, or in a combination with some lemmas). Thus we have to somehow transform the formula F to some corresponding $\mathcal{T}$-formula. We do it by generalisation: in $\mathcal{T}^e$-formula F, we generalise non-$\mathcal{T}$-terms (from outside in) by new variables in the following way: we substitute each non-$\mathcal{T}$-term of a type τ by a new variable of the same type and then take the universal closure of the formula F. Recall that all function symbols from Σ^e have either a type τ or a type of the form $\tau_{i_1} \times \tau_{i_2} \times \cdots \times \tau_{i_k} \rightarrow \tau$, where $\tau \in T$, so the formula F' obtained in such a manner is a $\mathcal{T}$-formula (with a possible exception of some redundant quantifiers of some types not in T). For instance, the extended Presburger arithmetic formula $\forall \alpha\ (min(\alpha) \leq max(\alpha))$ (where α has type `list of pnats`) can be transformed to $\forall min \forall max \forall \alpha\ (min \leq max)$ (where min and max are of type `pnats`). In deciding whether F' is a theorem, we use a decision procedure for $\mathcal{T}$ by first eliminating all new variables (either by using the decision procedure itself, or in a combination with some lemmas). It is preferable to eliminate variables obtained from the most complicated terms. Thus we have to introduce some total ordering on $\mathcal{T}^e$-terms. First, we define a function $|\ .\ |$ that maps the set of $\mathcal{T}^e$-terms into the set of natural numbers (it will correspond to the *size* of a term).

Definition 3. *If a $\mathcal{T}^e$-term* t *is a function symbol of some atomic type* τ $(\tau \in T)$ *or a variable, then* $|t| = 1$*. If a $\mathcal{T}^e$-term* t *is of the form* $f(t_1, t_2, \ldots, t_n)$*, then* $|t| = 1 + \sum_{i=1}^{n} |t_i|$*.*

Definition 4. *A $\mathcal{T}^e$-term* t_1 *is* heavier *than a $\mathcal{T}^e$-term* t_2 *iff*

- $|t_1| > |t_2|$ *or*
- $|t_1| = |t_2|$ *and a dominant symbol of* t_1 *comes later in the lexicographic ordering than a dominant symbol of* t_2 *or*

– $|t_1| = |t_2|$, $t_1 = f(t'_1, t'_2, \ldots, t'_n)$, $t_2 = f(t''_1, t''_2, \ldots, t''_n)$ *and there is a value* k $(1 \leq k \leq n)$ *such that* t'_i *and* t''_i *are identical terms (for* $i < k$*) and* t'_k *is heavier than* t''_k.

Definition 5. *If terms* t_1 *and* t_2 *are identical or* t_2 *is heavier than* t_1*, then* $t_1 \preceq t_2$.

Definition 6. *A* $\mathcal{T}^e$*-term* t *is* the heaviest non-$\mathcal{T}$-term *in some set* S *of* $\mathcal{T}^e$*-terms, if* t *is a non-*$\mathcal{T}$*-term and for every non-*$\mathcal{T}$*-term* t' *from* S *it holds* $t' \preceq t$.

The relation $\preceq$ defines a total ordering on the set of $\mathcal{T}^e$ terms and this ordering fulfils the following condition: provided finitely many variables and function symbols, for each term t there are finitely many terms t' such that $t' \preceq t$. (This condition is important for the termination of the method proposed.)

4 Simplification Procedures

In this section we describe four procedures for simplification (and for reducing the number of quantifiers) of $\mathcal{T}^e$-formula being proved. Each of them can be used in any simplification context, in combination with other components of a prover and, finally, they can together build an extended proof method for $\mathcal{T}$. All of the procedures are applicable just to $\mathcal{T}^e$-formulae.

We assumed that there is a decision procedure for the theory $\mathcal{T}$ based on the idea of successive elimination of quantifiers. Thus, let us suppose that there are available two procedures[3] dealing with $\mathcal{T}$-formulae — $\mathtt{DpElimOneQuantifier}_{\mathcal{T},A}$ and $\mathtt{DpGround}_{\mathcal{T}}$:

— let $\mathtt{DpElimOneQuantifier}_{\mathcal{T},A}$ be the procedure which transforms a given non-quantifier free $\mathcal{T}$-formula[4] F to prenex normal form, eliminates the innermost quantifier (and the corresponding variable) and returns a formula F' such that it holds $\mathcal{T} \vdash F$ iff $\mathcal{T} \vdash F'$ and the number of quantifiers in the formula F' is fewer (or one less) than the number of quantifiers in the formula F.

— let $\mathtt{DpGround}_{\mathcal{T}}$ be the procedure which for a given ground $\mathcal{T}$-formula F returns `true` if $\mathcal{T} \vdash F$ and returns `false` if $\mathcal{T} \not\vdash F$.

For many decidable theories there are such procedures ([13]).

For the following procedures, we restrict our consideration only to universally quantified conjectures and lemmas.

4.1 Decision Procedure for $\mathcal{T}$

Let the procedure $\mathtt{Dp}_{\mathcal{T},A}$ be defined in the following way: if a formula being proved is ground, then apply $\mathtt{DpGround}_{\mathcal{T}}$; otherwise, while a formula being proved is not ground, apply $\mathtt{DpElimOneQuantifier}_{\mathcal{T},A}$.

[3] Often these procedures can be also implemented in a flexible way: as the exhaustive applications of a series of rewrite rule sets (see [4])

[4] In F there could be some redundant quantifiers of some types not in T. This procedure ignores them (but does not eliminate them).

Note that $\mathtt{Dp}_{\mathcal{T},A}$ is a decision procedure for theory $\mathcal{T}$, i.e., $\mathtt{Dp}_{\mathcal{T},A}$ reduces a $\mathcal{T}$-formula F to `true` if $\mathcal{T} \vdash F$ and to `false` otherwise.

The decision procedure for the theory $\mathcal{T}$ defined in this way is more flexible, but usually somewhat slower than some more compact procedures which avoid unnecessary repetition of some algorithm steps (usually some unnecessary normalisations). Our extended proof method can also use some other decision procedure for the theory $\mathcal{T}$ instead of $\mathtt{Dp}_{\mathcal{T},A}$.

4.2 Elimination of Redundant Quantifier

The procedure `ElimRedundantQuantifier` for the elimination of a redundant quantifier is very simple: if there is a redundant quantifier (no matter of what type) in a $\mathcal{T}^e$-formula F being proved, eliminate it.

It is good if there are some rules for theory $\mathcal{T}$ such that we can use them to simplify the formula F before we try to apply `ElimRedundantQuantifier`. For instance, in PNA, we can reduce each atomic formula in such a way that it does not include two occurrences of the same term (for example, $k \leq max(a) + k$ can be rewritten to $0 \leq max(a)$).

4.3 Elimination of $\mathcal{T}$-Variables

Let a procedure $\mathtt{ElimOneVar}_{\mathcal{T},A}$ be defined in the following way: if F is a $\mathcal{T}^e$-formula and if there is a variable v which does not appear in non-$\mathcal{T}$-terms of F, then generalise all its non-$\mathcal{T}$-terms (from outside in), then use the procedure $\mathtt{DpElimOneQuantifier}_{\mathcal{T},A}$ to eliminate the variable v (and the corresponding quantifier) and then substitute new variables by the original generalised terms.[5] (Note that this procedure is applicable to $\mathcal{T}$-formulae, but it is only sensible to use this procedure for formulae which are $\mathcal{T}^e$-formulae and not $\mathcal{T}$-formulae.)

4.4 Elimination of Generalised Non-$\mathcal{T}$-Term Using a Lemma

A procedure $\mathtt{ElimGeneralisedTerm}_{\mathcal{T},A}$ is defined (only) for universally quantified $\mathcal{T}^e$-formulae. It is defined in the following way:

- if a formula F being proved is a $\mathcal{T}^e$-formula and is not a $\mathcal{T}$-formula, find the heaviest non-$\mathcal{T}$-term t in it;
- find a set $\mathcal{A}$ of all (different) atomic formulae (with their polarities) from F in which t occurs;

[5] Note that we consider only universally closed formulae, so we can freely reorder quantifiers (in a formula which is in prenex normal form) and the procedure $\mathtt{DpElimOneQuantifier}_{\mathcal{T},A}$ can be used to eliminate any of the quantifiers. In the general case, $\mathtt{ElimOneVar}_{\mathcal{T},A}$ would be defined in the following way: if F is $\mathcal{T}^e$-formula in prenex normal form and if there is a variable v with corresponding quantifier in the *innermost block of the same quantifiers* and which does not appear in non-$\mathcal{T}$-terms of F, then generalise all its non-$\mathcal{T}$-terms, use the procedure $\mathtt{DpElimOneQuantifier}_{\mathcal{T},A}$ to eliminate the variable v and then substitute new variables by the original generalised terms.

- For each member f of the set $\mathcal{A}$, try to find a lemma[6] $\forall x_1 \forall x_2 \ldots \forall x_n L_i$ such that:[7]
  - (i) if t' is the heaviest non-$\mathcal{T}$-term[8] in L_i, and if it occurs in an atomic formula l, there exists a (most general) substitution ϕ_i such that dominant predicates symbols of f and l match, f and l have the same polarity (in F and L_i respectively), terms t and t' occur in the same argument positions (in f and l respectively) and $t = t'\phi_i$.
  - (ii) all variables in L_i are substituted for some $\mathcal{T}$-terms in F (with corresponding types) by the substitution ϕ_i;
  - (iii) the heaviest non-$\mathcal{T}$-term in $L_i\phi_i$ is $t'\phi_i$.
- The new current goal is the formula[9] $F \vee \neg L_1\phi_1 \vee \cdots \vee \neg L_j\phi_j$; generalise all its non-$\mathcal{T}$-terms and then use the procedure $\texttt{DpElimOneQuantifier}_{\mathcal{T},A}$ to eliminate the variable v which corresponds to the term t; then substitute new variables for the original generalised terms; return the resulting formula as a current goal.

5 Extended Proof Method

We restrict our consideration only to universally quantified conjectures and lemmas. The extended proof method for the theory $\mathcal{T}$ (based on its decision procedure) we are proposing is defined in the following way:

(1) if possible (i.e., if there is a redundant quantifier), apply the procedure `ElimRedundantQuantifier` and go to step (1); otherwise, go to step (2).

(2) if possible (i.e., if a formula being proved is a $\mathcal{T}$-formula), use the procedure $\texttt{Dp}_{\mathcal{T},A}$:
- if it returns `true`, then the original conjecture is valid,
- if it returns `false` and
  - ∗ the step (4) has not been applied, then the original conjecture is invalid,

[6] We use instances of substitutivity axioms as lemmas, but we can also use them in the following way: if f is the dominant function symbol of t, for each pair $f(t_1, t_2, \ldots, t_m)$ and $f(u_1, u_2, \ldots, u_m)$ of different terms occurring in a formula being proved we use $t_1 = u_2 \wedge t_2 = u_2 \wedge \cdots \wedge t_m = u_m \rightarrow f(t_1, t_2, \ldots, t_m) =_\tau f(u_1, u_2, \ldots, u_m)$ as a lemma. This approach would have a wider scope, but would probably be less efficient.

[7] The motivation is that such lemmas contain information sufficient for proving the conjecture (although there are no guarantees of that kind).

[8] It is only sensible to search for lemmas which are $\mathcal{T}^e$ and are not $\mathcal{T}$-formulae; any lemma which is $\mathcal{T}$-formulae cannot contain any information that could not be derived by $\texttt{Dp}_{\mathcal{T},A}$; thus, we search just for lemma with non-$\mathcal{T}$-terms.

[9] Note that j does not have to be equal to the number of elements of $\mathcal{A}$: for some $\mathcal{T}^e$ conjectures to be proved no lemmas are required — for instance $\forall\alpha\ \forall k\ \ (max(\alpha) \leq k \vee max(\alpha) > k)$ can be proved without any lemma. Therefore, even if some of the lemmas with given properties are not found, it is still sensible to try to prove the conjecture by means of other lemmas and the theory $\mathcal{T}$ itself.

  - $*$ the step (4) has been applied, then return to that point and try to apply `ElimGeneralisedTerm`$_{\mathcal{T},A}$ in another way; if it is not possible, the extended proof method stops failing to prove or disprove the conjecture and returns the current goal.

otherwise, go to step (3).

(3) if possible (i.e., if a formula F being proved is a $\mathcal{T}^e$-formula and if there is a variable v which does not appear in non-$\mathcal{T}$-terms of F), apply the procedure `ElimOneVar`$_{\mathcal{T},A}$ and go to step (1); otherwise, go to step (4).

(4) if possible (i.e., if a formula being proved is a $\mathcal{T}^e$-formula and is not a $\mathcal{T}$-formula), apply the procedure `ElimGeneralisedTerm`$_{\mathcal{T},A}$ and go to step (1) (if `ElimGeneralisedTerm`$_{\mathcal{T},A}$ can be applied in more than one way, then keep this position for possible backtracks).

6 Worked Example

Here we consider an example from [3]. We use (PNA) as a theory $\mathcal{T}$ and Cooper's algorithm [6] as an algorithm A.[10] Let $\{i, j, k, l, \ldots\}$ be a set of variables of type `pnat` and $\{\alpha, \beta, \gamma, \ldots, \}$ be a set of variables of type `list of pnats`. We consider the conjecture:

$$\forall l\, \forall \alpha\, \forall k \quad (l \leq min(\alpha)\ \wedge\ 0 < k \ \rightarrow\ l < max(\alpha) + k)\ .$$

(1a) There are no redundant quantifiers in the conjecture, so go to step (2);

(2a) The conjecture is not a PNA-formula, so go to step (3);

(3a) There is a variable (k) which does not appear in not-PNA-terms ($min(\alpha)$ and $max(\alpha)$); generalize $min(\alpha)$ to min, $max(\alpha)$ to max and then use the procedure `ElimOneVar`$_{PNA,Cooper}$ to eliminate k from $\forall min \forall max \forall l \forall \alpha \forall k\ (l \leq min\ \wedge\ 0 < k\ \rightarrow\ l < max + k)$. We get $\forall min \forall max \forall l \forall \alpha\ (1 + min \leq l \vee l \leq max)$, and after substituting max by $max(\alpha)$ and min by $min(\alpha)$ we get $\forall l \forall \alpha\ (1 + min(\alpha) \leq l \vee l \leq max(\alpha))$.

(3b) There is a variable (l) which does not appear in not-PNA-terms ($min(\alpha)$ and $max(\alpha)$); generalize $min(\alpha)$ to min, $max(\alpha)$ to max and then use the procedure `ElimOneVar`$_{PNA,Cooper}$ to eliminate l from $\forall min \forall max \forall \alpha \forall l\ (1 + min \leq l \vee l \leq max)$. We get $\forall min \forall max \forall \alpha\ (min \leq max)$, and after substituting max by $max(\alpha)$ and min by $min(\alpha)$ we get $\forall \alpha\ (min(\alpha) \leq max(\alpha))$.

(4) The heaviest non-PNA-term in $\forall \alpha\ (min(\alpha) \leq max(\alpha))$ is $min(\alpha)$. Suppose that there is a lemma $L \equiv \forall \xi\ (min(\xi) \leq max(\xi))$ available. There is a substitution $\phi = \{\xi \mapsto \alpha\}$ such that $(min(\alpha)) = (min(\xi))\phi$. All preconditions of the procedure `ElimGeneralisedTerm`$_{PNA,Cooper}$ are fulfilled, so generalise all non-PNA-terms in the formula $\forall \alpha\ min(\alpha) \leq max(\alpha) \vee \neg(min(\alpha) \leq max(\alpha))$ — generalise $min(\alpha)$ to min and $max(\alpha)$ to max and then use the

[10] In the system described in [3] Hodes' algorithm [9] is used, which is incomplete and is sound only for universally closed PIA formulae. Experimental results show [10] that Cooper's decision procedure for PNA, despite its $2^{2^{2^n}}$ worst-case complexity, is, for practical purposes, no worse than one due to Hodes', so we use Cooper's procedure here.

procedure `DpElimOneQuantifier`$_{PNA,Cooper}$ to eliminate the variable min. We get $\forall max\ \forall\alpha(0 \leq 0)$, and after substituting max by $max(\alpha)$ we get $\forall\alpha\ (0 \leq 0)$.

(1b) We can eliminate the quantifier $\forall\alpha$ as α does not occur in $0 \leq 0$ (using `ElimRedundantQuantifier`) and we get $0 \leq 0$.

(2b) $0 \leq 0$ is the PNA-formula, so we can use the procedure `Dp`$_{PNA,Cooper}$ which returns `true`. Thus, the conjecture is valid.

7 Properties of Extended Proof Method

Termination. Each of the simplification procedures returns either a formula with fewer variables than the original formula or the number of variables are the same, but the heaviest non-$\mathcal{T}$-term in the original formula is heavier than the heaviest non-$\mathcal{T}$-term in the resulting formula. Since there are finitely many variables in the formula being proved and since for each non-$\mathcal{T}$-term there are finitely many non-$\mathcal{T}$-terms over that set of variables for which it is heavier than, the simplification procedures can be applied only finitely many times (provided a finite set of lemmas). Therefore, the described method terminates.

Soundness. We assume that the available procedures `DpElimOneQuantifier`$_{\mathcal{T},A}$ and `DpGround`$_{\mathcal{T}}$ are complete and sound. Besides, if a formula with non-$\mathcal{T}$-terms generalised to variables is proved valid, then the initial formula is valid too. Therefore, the procedures `Dp`$_{\mathcal{T},A}$, `ElimRedundantQuantifier` and `ElimOneVar`$_{\mathcal{T},A}$ are sound.

Let us prove that the procedure `ElimGeneralisedTerm`$_{\mathcal{T},A}$ is also sound. Let $(\forall\overrightarrow{x})F$ be a formula being proved, let $(\forall\overrightarrow{u})L$ be a lemma and let ϕ be a substitution that meets the conditions of the procedure `ElimGeneralisedTerm`$_{\mathcal{T},A}$ (we consider just the simple case with one lemma used; the general case can be handled similarly). From $\mathcal{T}^e \vdash (\forall\overrightarrow{u})L$ it follows[11] $\mathcal{T}^e \vdash (\forall\overrightarrow{x})L\phi$. Let us suppose that we proved $\forall\overrightarrow{x}(F \vee \neg L\phi)$. Thus, $\mathcal{T}^e \vdash (\forall\overrightarrow{x})L\phi$ and $\mathcal{T}^e \vdash \forall\overrightarrow{x}(F \vee \neg L\phi)$ imply $\mathcal{T}^e \vdash \forall\overrightarrow{x}(L\phi \wedge (F \vee \neg L\phi))$ and, further, $\mathcal{T}^e \vdash \forall\overrightarrow{x}(L\phi \wedge F)$. Finally, it holds $\mathcal{T}^e \vdash \forall\overrightarrow{x}F$, i.e., $(\forall\overrightarrow{x})F$ is valid, which is what we wanted to prove.

(For the soundness of the presented extended proof method, it is sufficient that procedures `DpElimOneQuantifier`$_{\mathcal{T},A}$ and `DpGround`$_{\mathcal{T}}$ are sound. For instance, Hodes' algorithm can be used for the PNA procedure which would be sound (and incomplete) for universally closed formulae.)

(In)completeness. We do not make any claims about the completeness of our extended proof method: we do not see this as a severe weakness since we intend to exploit this approach in undecidable theories — we are trying to build a proof method that is successful outside the realm of a decision procedure. In a certain sense, our approach is complete since it is strictly an extension of some underlying decision procedure: those formula falling with the scope of that procedure will be decided correctly.

[11] We assume that $(\forall\overrightarrow{u})L$ is proved valid in $\mathcal{T}^e$.

Efficiency. In the proposed extended proof method there are some unnecessary repetitions due to the flexible combination of independent modules. However, these steps (generalisation of non-$\mathcal{T}$-terms, substitutions, some normalisations etc) can usually be executed in linear time on the size of a current goal and therefore do not significantly affect the efficiency of the system. Moreover, usually all steps of the method can be executed in linear time on the size of the formula being proved. Thus, the complexity of the presented method is dominated by the complexity of the procedures $\texttt{DpElimOneQuantifier}_{\mathcal{T},A}$ and $\texttt{DpGround}_{\mathcal{T}}$.

Besides, although the proposed method may seem a bit complicated, it is generally intended to be used for some simpler conjectures, so the described steps would be simple and fast. Additionally, we could restrict the use of the method just to some conjectures for which it is likely that the method will be successful (for this restriction we could use different heuristic scores or stochastic scores) and we could use some other techniques (e.g. induction) in other cases.

Flexibility. There is a trade-off between generality and efficiency in building an extended proof method. The proposed method is built by combining independent modules which could decrease efficiency, though hopefully not significantly. On the other hand however, the proposed method has a high degree of generality and flexibility: in a uniform way it can be used in different theorem provers, for different theories and for different decision procedures for these theories. It also does not require any specific data structures (for example, a database of polynomials). We claim that that potential losses in efficiency are dominated by these advantages.

8 Implementation and Results

We have made a preliminary implementation of the proposed extended proof method within the Clam system [5]. We have implemented procedures for PIA based on Hodes' algorithm[12] and for PIA and PNA based on Cooper's algorithm.[13] These implementations are also flexible and based on the idea of the exhaustive applications of a series or rewrite rule sets (see [4]). This, preliminary version of the extended proof method successfully proved a number of conjectures. Some results (obtained on examples from [3]) are given in Table 1.[14] We applied the extended proof method with three different algorithms for Presburger arithmetic (so, the variable l is of type `integer` in 1a and 1b and of type `pnat` in 1c etc.). The lemmas are formulae valid over natural numbers (but can be used in an adapted form with procedures working over the integers). It can be seen from the table that all three variants had problems with the third conjecture (the

[12] Hodes' procedure [9] is the decision procedure for PRA and the algorithm for PIA based on it is incomplete and is sound only for universally closed PIA formulae.

[13] Cooper's procedure [6] is the decision procedure for PIA, but can be adapted to the decision procedure for PNA.

[14] The extended proof method is implemented in the Clam proof-planning system under SWI Prolog. Tests are made on a PC486 16Mb, running under Linux. CPU time is given in seconds.

#	$\mathcal{T}$	A	lemmas	CPU time (s)
1	$\forall l \forall \alpha \forall k\ (l \leq min(\alpha) \wedge 0 < k \rightarrow l < max(\alpha) + k)$			
1a	PIA	Hodes'	$\forall \xi\ (min(\xi) \leq max(\xi))$	3.56
1b	PIA	Cooper's	$\forall \xi\ (min(\xi) \leq max(\xi))$	5.32
1c	PNA	Cooper's	$\forall \xi\ (min(\xi) \leq max(\xi))$	4.11
2	$\forall lp \forall lt \forall i \forall pat \forall c\ lp + lt \leq maxint \wedge i \leq lt \rightarrow i + delta(pat, lp, c) \leq maxint$			
2a	PIA	Hodes'	$\forall x \forall y \forall z\ delta(x, y, z) \leq y$	1.74
2b	PIA	Cooper's	$\forall x \forall y \forall z\ delta(x, y, z) \leq y$	2.52
2c	PNA	Cooper's	$\forall x \forall y \forall z\ delta(x, y, z) \leq y$	12.47
3	$\forall a \forall b \forall c ms(c) + ms(a)^2 + ms(b)^2 < ms(c) + ms(b)^2 + 2 \cdot ms(a)^2 \cdot ms(b) + ms(a)^4$			
3a	PIA	Hodes'	$\forall x\ 0 < ms(x)$, $\forall i \forall j\ 0 < i \rightarrow j \leq i \cdot j$	32.35
3b	PIA	Cooper's	$\forall x\ 0 < ms(x)$, $\forall i \forall j\ 0 < i \rightarrow j \leq i \cdot j$	108.25
3c	PNA	Cooper's	$\forall x\ 0 < ms(x)$, $\forall i \forall j\ 0 < i \rightarrow j \leq i \cdot j$	?
4	$\forall a \forall b \forall c ms(c) + ms(a)^2 + ms(b)^2 < ms(c) + ms(b)^2 + 2 \cdot ms(a)^2 \cdot ms(b) + ms(a)^4$			
4a	PIA	Hodes'	$\forall i \forall j\ j \leq ms(i) \cdot j$	9.77
4b	PIA	Cooper's	$\forall i \forall j\ j \leq ms(i) \cdot j$	35.49
4c	PNA	Cooper's	$\forall i \forall j\ j \leq ms(i) \cdot j$	3.41
5	$\forall k \forall l\ 0 < k \wedge 0 < l \wedge 2 \cdot k + 1 \leq 2 \cdot l \rightarrow 2 \cdot k + 2 \leq 2 \cdot l$			
5a	PIA	Hodes'		/
5b	PIA	Cooper's		11.27
5c	PNA	Cooper's		1.14

Table 1. Results of the preliminary implementation of the extended proof method

variant with Cooper's algorithm for PNA even ran out of the standard stack size in our system). In this example, lemmas have to be invoked six times and this leads to very complex intermediate formulae. However, if we change two lemmas from the third example with one lemma that is sufficient for a proof (the fourth example), the situation is changed and the system is more efficient. The explanation is the following: the facts $0 < ms(a)$ and $0 < ms(b)$ could not have been used while the terms $ms(a)$ and $ms(b)$ were not the heaviest non-$\mathcal{T}$-terms, so the intermediate formulae were very complex (they involved a large number of formulae $0 < ms(a)$ and $0 < ms(b)$). This was not a problem in the fourth example. This problem can be avoided in some cases: if the lemma we use is of the form $\forall \overrightarrow{x}(H_1 \wedge \cdots \wedge H_k \rightarrow C)$ then, after instantiation, we can try to prove the atomic formulae H_i ($i = 1, \ldots, k$) separately and then, in the main proof we can use just the atomic formula C (the system [3] uses lemmas in this way). This approach can significantly increase the performances of the system in some cases (for instance, for the formula $\forall a \forall b \forall c\, ms(c) + ms(a)^2 + ms(b)^2 < ms(c) + ms(b)^2 + 2 \cdot ms(a)^2 \cdot ms(b) + ms(a)^4$, a speed-up similar to that obtained in the fourth example would be obtained).

The fourth example also illustrates the fact that Cooper's procedure for PNA can be much more efficient than the one for PIA or Hodes' procedure for PIA. Thus, it seems that it is sensible to have all these procedures available (while their ordering could be based on some heuristic scores and adjusted dynamically).

The fifth conjecture is invalid over reals and cannot be proved by Hodes' algorithm (and hence by the system presented in [3]), but can be proved by Cooper's algorithms for PIA and PNA. Thus, this example demonstrates the utility of having available different procedures for 'Presburger arithmetic'.

9 Future Work

Some of the ways in which the proposed method could be improved by making it more general are as follows:

- deal with predicates from $\Pi^e \setminus \Pi$; this could be done by using lemmas and generalising to variables of type `truth` (in a very similar manner we use for atomic formulae with non-$\mathcal{T}$-terms);
- use substitutivity axioms not only as lemmas;
- broaded scope to non-universally closed formulae;
- use definitions or lemmas about $\mathcal{T}^e$ functions and predicates expressed in terms of the theory $\mathcal{T}$ (e.g., we can always rewrite $double(x)$ to $2x$ by using a lemma $\forall x\ (double(x) =_{\texttt{pnat}} 2x)$ and we can always rewrite $x \neq y$ to $x < y \vee y < x$ by using a lemma $\forall x\ \forall y\ (x \neq y \leftrightarrow x < y \vee y < x)$);

Some of the ways in which the efficiency of the proposed method could be improved are the following:

- make the condition (i) in 4.4 more restricted — try to match whole atomic formulae f and l (not just the heaviest non-$\mathcal{T}$-terms); this approach would be more efficient for come conjectures, but would have smaller scope: e.g., within this approach $\forall\alpha\forall k\ min(\alpha) \leq max(\alpha) + k$ could not be proved using the lemma $\forall\xi(min(x) \leq max(x))$;
- preprocess lemmas or use lemmas having some fixed structure (e.g., those of the form $\forall\overrightarrow{x}(H_1 \wedge \cdots \wedge H_k \to C)$);

In future work we will investigate different ways for improving the presented extended proof method. We intend to make a more systematic study in order to find an optimum between generality and efficiency. In future work we also intend to explore possible combinations of this approach with other ones.

10 Related Work

In the last two decades a lot of effort has been invested into efficient and flexible integration of decision procedures (in particular those for arithmetic) into general-purpose theorem provers or domain specific systems. Our method is mostly related to a procedure for linear arithmetic integrated within Boyer and Moore's NQTHM [3]. This system is rather efficient, but involves a lot of special data structures and the description of the procedure is often given in terms of these special data-structures rather than in terms of their logical meaning. Besides, this system is adjusted for Hodes' algorithm and cannot be used on some other theory or some other algorithm. It is also incomplete for PNA. Our approach is a kind of rational reconstruction of Boyer and Moore's linear arithmetic procedure and is also its generalisation in some aspects. There are some losses in efficiency, but gains are in generality and flexibility.

Several systems are based on [16], including Stanford Pascal Verifier [14] and STeP [15]. In this approach, decision procedures for disjoint theories are combined by abstracting terms which fall outside a certain theory and by propagating deduced equalities from one theory to another. Several other systems are based on [19], including PVS [17] and EHDM [8]. In this approach, an efficient congruence closure procedure is used to handle combinations of ground equalities involving uninterpreted function symbols, linear arithmetic and arrays. Congruence closure is used in combination with decision procedures (solvers) for specific theories (for instance, with a solver for Presburger inequalities). There is an analysis of Shostak's algorithm and a partial analysis of an algorithm due to Nelson and Oppen in [7]. These approaches focus on combinations of (disjoint) theories in contrast to extensions of theories (which are not disjoint, but move outside some syntactically defined class). These systems are rather efficient over their intended domains, however, the method presented in this paper is more syntactical in its nature and more suited to a proof-planning environment (such as embodied in Clam [5]). There is also work on incorporating an arithmetic decision procedure into a rewrite based prover [11] and work in equational reasoning in resolution and paramodulation based provers [2,1].

11 Conclusions

We have presented a method for the flexible integration certain decision procedures into theorem provers. It is partly based on [3], but is more general and more flexible. The method can be used in different theorem provers, for different theories and for different decision procedures for these theories. Specific knowledge is encapsulated in smaller submodules and decision procedures can be simply 'plugged-in' to the system. This framework is well-suited to the proof-planning paradigm. We have made a preliminary implementation within the Clam system and the results are most encouraging. In future work we propose to investigate different refinements of the method (outlined in section 9), including possible combination with other approaches.

References

1. L. Bachmair and H. Ganzinger. Strict basic superposition. In Kirchner and Kirchner [12], pages 160–174.
2. L. Bachmair, H. Ganzinger, and A. Voronkov. Elimination of equality via transformation with ordering constraints. In Kirchner and Kirchner [12], pages 175–190.
3. R. S. Boyer and J S. Moore. Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic. In J. E. Hayes, J. Richards, and D. Michie, editors, *Machine Intelligence 11*, pages 83–124, 1988.
4. A. Bundy. The use of proof plans for normalization. In R. S. Boyer, editor, *Essays in Honor of Woody Bledsoe*, pages 149–166. Kluwer, 1991.

5. A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449.
6. D. C. Cooper. Theorem proving in arithmetic without multiplication. In B. Meltzer and D. Michie, editors, *Machine Intelligence 7*, pages 91–99. Edinburgh University Press, 1972.
7. D. Cyrluk, P. Lincoln, and N. Shankar. On Shostak's decision procedure for combinations of theories. In M. McRobbie and J. Slaney, editors, *13th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, Vol. 1104, New Brunswick, NJ, USA, 1996. Springer-Verlag.
8. User guide for the EHDM specification language and verification system, version 6.1. Technical report, SRI Computer Science Laboratory, 1993.
9. L. Hodes. Solving problems by formula manipulation in logic and linear inequalities. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 553–559, Imperial College, London, England, 1971. The British Computer Society.
10. P. Janičić, I. Green, and A. Bundy. A comparison of decision procedures in Presburger arithmetic. In R. Tošić and Z. Budimac, editors, *Proceedings of the VIII Conference on Logic and Computer Science (LIRA '97)*, pages 91–101, Novi Sad, Yugoslavia, September 1–4 1997. University of Novi Sad.
11. D. Kapur and X. Nie. Reasoning about numbers in Tecton. In *Proceedings of 8th International Symposium on Methodologies for Intelligent Systems*, Charlotte, North Carolina, USA, October 1994.
12. C. Kirchner and H. Kirchner, editors. *15th International Conference on Automated Deduction*, Lindau, Germany, July 1998.
13. G. Kreisel and J. L. Krivine. *Elements of mathematical logic: Model theory.* North Holland, Amsterdam, 1967.
14. D. C. Luckham, S. M. German, F. W. von Henke, R. A. Karp, P. W. Milne, D. C. Oppen, W. Polak, and W. L. Scherlis. Stanford Pascal verifier user manual. CSD Report STAN-CS-79-731, Stanford University, 1979.
15. Z. Manna et al. STeP: The Stanford Temporal Prover. Technical Report STAN-CS-TR;94-1518, Stanford University, 1994.
16. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, October 1979.
17. S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. A. Henzinger, editors, *Proceedings of the 1996 Conference on Computer-Aided Verification*, number 1102 in LNCS, pages 411–414, New Brunswick, New Jersey, U. S. A., 1996. Springer-Verlag.
18. M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Sprawozdanie z I Kongresu metematyków slowiańskich, Warszawa 1929*, pages 92–101, 395. Warsaw, 1930. Annotated English version also available [20].
19. R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, January 1984.
20. R. Stansifer. Presburger's article on integer arithmetic: Remarks and translation. Technical Report TR 84-639, Department of Computer Science, Cornell University, September 1984.

Presenting Proofs in a Human-Oriented Way

Helmut Horacek

Universität des Saarlandes
FB 14 Informatik
Postfach 1150, D-66041 Saarbrücken, Germany
horacek@cs.uni-sb.de

Abstract. Presenting machine-generated proofs in terms adequate to the needs of a human audience is a serious challenge. One salient property of mathematical proofs as typically found in textbooks is that lines of reasoning are expressed in a rather condensed form by leaving out elementary and easily inferable, but logically necessary inference steps, while explaining involved ones in more detail. To date, automated proof presentation techniques are not able to deal with this issue in an adequate manner. Addressing this problem in a principled way, we describe an approach that successively enhances a logically self-contained proof at the assertion level through communicatively justified modifications of the original line of reasoning. These enhancements include expansion of involved theorem applications, omission of trivial justifications, compactification of intermediate inference steps, and broadening the scope of justifications to support focused argumentation, as in chains of inequations. Through incorporating these measurements, many proofs are presented in a shorter and better understandable fashion than in previous approaches.

1 Introduction

Presenting machine-generated proofs in terms adequate to the needs of a human audience is a serious challenge. Mathematical proofs as typically found in textbooks fundamentally differ from machine-generated proofs: they are well-structured and methodologically organized, they entail focused chains of inferences such as series of inequations, and they express lines of reasoning in a condensed form by leaving out elementary and easily inferable, but logically necessary inference steps, while explaining involved ones in more detail. The last-mentioned property is perhaps one of the most salient differences to machine-generated proofs. By far, current proof presentation techniques are not able to deal with this issue in an adequate manner.

In order to produce presentations that better resemble those found in textbooks, we take into account inferential capabilities and memory limitations of humans. We do this by introducing communicatively justified and in parts addressee dependent proof graph enhancements into a logically self-contained proof at the assertion level, as introduced by Huang [13]. This technique is inspired by cognitive models of presenting texts with inference-rich contents [10, 11], and it is motivated by insights gained through empirical studies [16, 22, 23]. Proof graph enhancements include expansion of involved theorem applications, omission of trivial justifications, compactification of intermediate inference steps, and broadening the scope of justifications to support focused argumentation, as in chains of inequations. The resulting structure is verbalized by the natural language generator PROVERB [15].

This paper is organized as follows: First, we review presentation techniques for proofs and for other domains with inference-rich contents. Then we motivate our approach empirically. We follow by outlining the intuitions behind and its basic functionality. Then we describe the formalization of presentation rules, a central part in our method. We illustrate the improvements obtained by presenting the solution to the relatively complex Steamroller problem [21]. Finally, we discuss assumptions incorporated in our method, computational efforts involved, and potential extensions.

2 Approaches to Proof Presentation in Natural Language

The first attempt to express machine-found proofs in natural language has been undertaken by Chester [2] in his system EXPOUND. A more recent system is THINKER [5], which can present natural deduction proofs in varying styles. Apart from other presentation and structuring methods, ILF [4] is able to produce proofs in natural language. Embedded in a proof development environment, CtCoq [3] generates compactified proofs in natural language, by composing text patterns associated with proof object types. Finally, the system developed in our own research group, PROVERB [15], expresses machine-found proofs abstracted to the assertional level [13] and applies linguistically motivated techniques for text planning, generating referring expressions, and aggregation of propositions with common elements.

In order to produce reasonable proof presentations, many systems describe some complex inference steps very densely, and they leave certain classes of proof steps implicit in their output, e.g., CtCoq abstracts from intermediate inference steps recoverable from inductive definitions, and PROVERB omits instantiations of axioms. However, leaving out information on the basis of purely *syntactic* criteria, as this has been done so far, easily leads to incoherent and hardly understandable text portions. In order to get control over the inferability and comprehensibility in presenting inference steps, an explicit model is required which incorporates semantic and pragmatic aspects of communication; first steps towards such a model have been undertaken in [7]. In this paper, we extend this approach significantly.

Unlike in mathematical texts, the significance of inferability is less pronounced in other domains. Nevertheless, a few methods in the field of natural language generation try to anticipate a user's likely inferences and to exploit them by conveying information indirectly. Zukerman and McConachy [24] select a subset of content specifications for presentation, thereby exploiting inferable relations expressed in a taxonomic hierarchy. Green and Carberry [8] aim at the generation of indirect answers to accomplish complementary discourse goals by modeling potential obstacles that prevent intended achievements. Horacek [10] attempts to capture inference relations between generic and referential pieces of knowledge through rules expressing aspects of conversational implicature [8]. These rules put a good deal of effort in modeling relevance-driven expectations and communicative preferences typical of real world but not of abstract domains. The basic mechanism, however, is relevant to proof presentation. In [11], we have proposed a much simpler version, elaborated for everyday discourse, threreby enhancing the coverage to chains of inferences, and supporting argumentation structure reorganization, which we adopt here for our purposes.

3 Issues in Presenting Proofs in a Human-Oriented Way

Issues in presenting deductive proofs, as a special case of presenting argumentative discourse, have attracted a lot of attention in the fields of psychology, linguistics, and computer science. Central insights relevant to deductive argumentation are:

- Logical consequences of certain kinds of information are preferably conveyed implicitly through exploiting the discourse context and default expectations.
- Human performance in comprehending deductive syllogisms varies significantly from one syllogism to another.
- Empirically observed successful discourse strategies reintroduce logically redundant information to support the addressee's attention in involved cases.

The study in [22] demonstrates that humans easily uncover missing pieces of information left implicit in discourse, provided this information conforms to their expectations in the given context. Similarly to the expectations examined in that study, which occur frequently in everyday conversations, a number of elementary and very common inferences are typically left implicit in mathematical texts, too, including straightforward instantiations, generalizations, and associations justified by domain knowledge. Moreover, all sorts of texts favor the organization of the material to convey in coherent and focused argumentation sequences without side-steps, which is in contrast to the nested structures in automatically generated proofs.

Another presentation aspect is addressed by studies on human comprehension of deductive syllogisms (see the summary in [16]). These studies have unveiled considerable performance differences among individual syllogisms (in one experiment, subjects made 91% correct conclusions for modus ponens, 64% for modus tollens, 48% for affirmative disjunction, and 30% for negative disjunction). This effect is explained by the number of mental models to be maintained in each case – keeping only one model in mind, as for modus ponens, is significantly easier than the two models needed for the other syllogisms (or three, for inclusive disjunction).

Finally, the elaborate essay in [23] presents a number of hypotheses about the impacts human resource limits in attentional capacity and in inferential capacity have on dialog strategies. These hypotheses are acquired from extensive empirical analysis of naturally occurring dialogs and, to a certain extent, statistically confirmed. Two of them are the following: (1) an increasing number of logically redundant assertions to make an inference explicit are made, in dependency of how hard and important an inference is (modus tollens being an example for a hard inference), and (2) the need to make discourse objects salient prior to referring to them, if they are not salient yet.

In the following, we demonstrate that these crucial issues in presenting deductive reasoning are insufficiently captured by current techniques. By some typical examples, we discuss deficits in the automatically produced proof presentations, necessary changes, and suitable measurements to achieve these improvements.

Consider the portions of straightforwardly presented proofs produced by an earlier version of PROVERB, (texts (1) to (3) in Figure 1), each of which can be improved significantly, as demonstrated by texts (1') to (3'), correspondingly. Texts (1) and (2) should be presented more concisely, while parts of text (3) require more explanation. In (1'), the addressees' knowledge about definitions (here, concerning equivalence relations), and their capabilities to mentally perform some sort of simple inference steps such as 'and'-eliminations and elementary substitutions are exploited. In (2'),

(1) „Let ρ be an equivalence relation. Therefore we have ρ is reflexive, we have ρ is symmetric, and we have ρ is transitive. Then we have ρ is symmetric and we have ρ is reflexive. Then $\forall x\colon x \rho x$. Thus we have $h_0 y_0 \rho h_0 y_0$. ...“

(1') „Let ρ be an equivalence relation. Thus we have $h_0 y_0 \rho h_0 y_0$. ...“

(2) „Let $1 < a$. Since lemma 1.10 holds, $0 < a^{-1}$. Since $1 < a$ holds and '<' is monotone, $1a^{-1} < aa^{-1}$ holds. $a^{-1} < aa^{-1}$ because of the unit element of K. $aa^{-1} = 1$ because of the inverse element of K for $a \neq 0$. Thus $a^{-1} < 1$.“

(2') „Let $1 < a$. Since lemma 1.10 holds, $0 < a^{-1}$. Then $0 < a^{-1} = 1a^{-1} < aa^{-1} = 1$.“

(3) „Let ρ be a transitive relation and let $\neg(a \rho b)$. Let us assume that $c \rho b$. Hence we have $\neg(a \rho c)$.“

(3') „Let ρ be a transitive relation and let $\neg(a \rho b)$. Let us assume that $c \rho b$. Since ρ is transitive, $\neg(a \rho b)$ implies that $\neg(a \rho c)$ or $\neg(c \rho b)$ holds. Since we have $\neg(a \rho b)$ and $c \rho b$, $\neg(a \rho c)$ follows.“

Fig. 1. Straightforwardly presented proof portions and suitable improvements

the compact notation format for series of inequations is used. In order for this presentation to be understood by the intended audience, the addressees must be acquainted with the axioms applied (here: monotony, unit and inverse elements), and capable of mentally inferring the exact place where they apply. In (3'), the involved application of the transitivity axiom is exposed more explicitly through separating the descriptions of the instantiation of the theorem (in reversed direction as a modus tollens) from the disjunction elimination inference, thereby reintroducing the facts not mentioned in immediately preceding utterance parts. Altogether, these examples show some crucial deficits in current proof presentation techniques:

- A large number of easily inferable inference steps is expressed explicitly.
- Fluent presentations of homogeneous reasoning lines are insufficiently used.
- Involved inferences, though hard to understand, are presented in single shots.

The first deficit suggests the omission of contextually inferable elements in the proof graph, the second one calls for regrouping portions of the underlying argumentation structure, and the third one demands the expansion of compound inference steps into simpler parts. These aims are ambitious, and their pursual requires the development of new, well-informed techniques, to which we contribute in the following sections.

4 Obtaining Concise and Informative Proof Presentations

In order to obtain presentations similar to (1'), (2'), and (3') in Figure 1, we propose the application of an optimization process that enhances an automatically generated proof at the assertional level. Through this process, pragmatically motivated expansions, omissions, short-cuts, and reorderings are introduced, and the audience is assumed to be able to mentally reconstruct the details omitted with reasonable effort. In a nutshell, the modified proof graph is built through four subprocesses:

1. Building *expansions*
   Compound assertion level steps are expanded into elementary applications of deductive syllogisms, while marking the original larger steps as *summaries*.
2. Introducing *omissions* and *short-cuts*
   Shorter lines of reasoning are introduced by skipping individual reasoning steps, through omitting justifications (marked as *inferable*) and intermediate reasoning steps (marking the 'indirect' justifications as *short-cuts*).
3. Reorganizing the *argumentation structure*
   In order to encapsulate subproofs for building lemmata, embedded justifications are regrouped by copying them to a place in the proof graph with a more general scope. In their original place, they are marked as *co-references*.
4. *Selecting* the information to convey
   Content selection is done by: avoiding summaries, omitting inferables, pursuing the closest short-cuts, and excluding justifications of co-references.

The selected path in the proof graph is ultimately expressed in natural language by PROVERB. In the following, we explain subprocesses 1 to 3 in more detail.

The purpose underlying the expansion of assertion levels steps is to decompose presentations of complex theorem applications or involved applications of standard theorems into easier comprehensible pieces. This operation is motivated by performance difficulties humans typically have in comparable discourse situations. At first, assertion level steps are completely expanded to the natural deduction (ND) level according to the method described in [12]. Thereafter, a partial recomposition of ND-steps into inference steps encapsulating the harder comprehensible deductive syllogisms, modus tollens and disjunction elimination steps, is performed, in case the sequence of ND rules in the entire assertion level step contains more than one of these. To do this, the sequence of ND rules is broken after each but the last occurence of a modus tollens or disjunction elimination, and the resulting subsequences of ND-steps are composed into a sequence of reasoning steps at some sort of *partial assertion* level. This sequence is then inserted in the proof graph as a potential substitute for the original assertion level step, which is marked as a *summary*. An example for such an expansion and partial recomposition is shown in Figure 2 ($\forall$E, $\Rightarrow$ E, $_\vee$E, and NR stand for the ND rules, forall-elimination, implication elimination, disjunction elimination, and natural rewrite, respectively). If $b \rho c$ and $\neg (a \rho c)$ hold for a transitive relation ρ, $\neg (a \rho b)$ is derivable by a single assertion level step ((1) in Figure 2). Through expansion to the ND level ((2) in Figure 2) and recomposition encompassing deductive syllogisms ((3) in Figure 2), the modus tollens inference step "$\neg (a \rho c)$ implies $\neg (a \rho b)$ or $\neg (b \rho c)$" is separated from the disjunction elimination "Thus, $b \rho c$ yields $\neg (a \rho b)$". Note that, in contrast to modus tollens, modus ponens would be composed with disjunction elimination into a single step at the partial assertion level. For example, an application of the rule $\forall$n: $((n \in \mathbb{Z}\backslash\{0\}) \Rightarrow ((n > 0) \vee (n < 0)))$ to the premises $(n \in \mathbb{Z}\backslash\{0\})$ and $\neg (n < 0)$ is presented in one step.

Unlike expanding summaries, creating omissions, short-cuts, and reorganizations is driven by communicatively motivated *presentation* rules. They express aspects of human reasoning capabilities with regard to contextually motivated inferability of pieces of information on the basis of explicitly mentioned facts and relevant background knowledge [9]. These rules provide an interface to stored assumptions about the intended audience. They describe the following sorts of situations:

$$(1)\quad \dfrac{\forall x,y,z: ((x \rho y \wedge y \rho z) \Rightarrow x \rho z)\ {}_{\text{Assertion}},\ \neg (a \rho c),\ b \rho c}{\neg (a \rho b)}$$

$$(3)\quad \dfrac{\dfrac{\forall x,y,z: ((x \rho y \wedge y \rho z) \Rightarrow x \rho z)\ {}_{\text{Modus tollens}},\ \neg (a \rho c)}{\neg (a \rho b) \vee \neg (b \rho c)}\ {}_{\vee\text{E}},\ b \rho c}{\neg (a \rho b)}$$

$$(2)\quad \dfrac{\dfrac{\dfrac{\dfrac{\forall x,y,z: ((x \rho y \wedge y \rho z) \Rightarrow x \rho z)\ \forall_{\text{E}}}{(a \rho b \wedge b \rho c) \Rightarrow a \rho c}\ {}_{\Rightarrow\text{E}},\ \neg (a \rho c)}{\neg (a \rho b \wedge b \rho c)}\ {}_{\text{NR}}}{\neg (a \rho b) \vee \neg (b \rho c)}\ {}_{\vee\text{E}},\ b \rho c}{\neg (a \rho b)}$$

Fig. 2. An involved assertion level inference at several degrees of abstraction

Cut-prop-rule – omission of a *proposition* (premise) appearing as a justification
Cut-rule-rule – omission of a *rule* (axiom instance) appearing as a justification
Compactification-rule – short-cut by omitting an *intermediate* inference step
Restructuring-rule – giving a justification more general scope in the proof graph

The first three rules, the *reduction* rules, aim at omitting a justification that the audience is considered to be able to infer from the remaining justifications of the same line of the proof, or even at omitting an entire assertion level step that is considered inferable from the adjacent inference steps. In order for these rules to apply successfully, presentation preferences and conditions about the addressees' knowledge and inferential capabilities are checked, which is explained in the following section.

The functionality of the reduction rules can be explained by a simple example. If trivial facts, such as $0 < 1$, or axioms assumed to be known to the audience, such as *transitivity*, appear in the set of justifications of some inference step, they are marked as *inferable* ($0 < 1$ through the *Cut-prop*-rule, and *transitivity* through the *Cut-rule*-rule). Consequently, the derivation of $0 < a$ can simply be explained by $1 < a$ to an informed audience. Moreover, single facts appearing as the only non-inferable justification are candidates for being omitted through applying the *Compactification*-rule. If, for instance, $0 < a$ is the only non-inferable justification of $0 \neq a$, and $0 < a$, in turn, has only one non-inferable justification, $1 < a$, the coherence maintaining similarity between $0 < a$ and $1 < a$ permits omitting $0 < a$ in the argumentative chain. Altogether, $0 \neq a$ can be explained concisely by $1 < a$ to an informed audience.

The purpose of the *Restructuring*-rule is to obtain focused lines of reasoning, by making nested justifications of some proof line self-contained. This requirement is motivated by the applicability of special presentation modes, such as building chains of inequations, and it is also suitable for multiple referred justifications. Applying this rule causes a premise P, which is required to be self-contained in some proof line L_1 to be inserted as an additional justification in a proof line L_2 justified directly or indirectly by L_1. Moreover, P is marked as *co-reference* in the scope of L_1.

The presentation rules are matched against proof lines in three processing cycles. In each cycle, the proof graph is traversed by starting from its leaf nodes and successively continuing to the root node, without back-tracking (that is, some sort of inverse depth-first search is invoked): In cycle one, the *Cut-prop*-rule and the *Cut-rule*-rule are applied, marking locally inferable justifications. In cycle two, the *Compactification*-rule is invoked, adding alternative justifications through short-cuts, on the basis of the inferables, and in cycle (3) the *Restructuring*-rule is applied. This order takes into account dependencies among the rules. It is also reasonably efficient, since only short-cuts require processing alternative lines of reasoning.

5 Formalization of the Presentation Rules

In this section, we illustrate the formalization of the presentation rules, which constitute the key part of our method. These rules are adopted from the domain-independent formulation given in [11], where they operate on rhetorical structure theory (RST) [17] trees. RST trees are mostly binary trees, where each junction is associated with semantics through a rhetorical relation. Despite different organization principles, there is an easy mapping from RST trees onto proof graphs. For our purposes, we reformulate the presentation rules for proof graphs, and we incorporate interpretations of the parts expressing presentation knowledge that are specific to mathematical proofs. In the following, we explain the interfaces of the presentation rules to the proof graph and to presentation-specific knowledge. Then we give the formalization of these rules in detail, thereby illustrating their effect by a typical example. Finally, we discuss the domain-specific formalization of presentation knowledge.

For interfacing the proof graph, we assume the availability of information for each proof line as in the Ω-system [1], including the formula concluded and a set of justifications pointing to other lines. We slightly extend this representation here:

- Individuals and sets of justifications may be associated with *marks* that express one of the special states *Chainable* for proof lines (introduced by the method in [6], aiming at focused argumentation), *Summary* for justification sets, and *Inferable*, *Short-cut*, and *Co-reference* for individual justifications.
- We enhance the implicitly conjoined set of justifications to disjunctions of such sets for alternative justifications (including summaries and short-cuts).

The presentation rules make use of several functions operating on the lines of the proof graph by providing access to their elements (see Table 1). The MAIN-TERM in a set of justifications can be viewed as a premise, too, but its contextual inferability by humans differs from that of ordinary premises. In the Ω-system, this term is always put in the same prominent position by the methods creating the proof lines, so that it can easily be accessed. In addition to these proof graph access functions, the presentation rules contain some predicates that express mental capabilities of the audience, namely the *attentional state* (AWARE-OF) and *inferential skills* (ABLE-INFER and COHERENT), as well as addressee-independent presentation preferences (NO-EXTEND). Finally, we use the non-deterministic choice-function ONE-OF in order to select list elements, which is applied to proof lines, justifications, and premises.

CONCLUSION(L)	the formula which is derived through proof line L
EXPLAINS(L)	the set of proof lines justified by line L
JUSTIFICATIONS(L)	the alternative justification sets for the conclusion of line L
PREMISES(L,J)	the set of ordinary premises in the justification set J of line L
MAIN-TERM(L,J)	the main term among the justifications in set J in line L
GENERIC(L)	the generic form of an axiom if line L is an instance of it
AXIOM(L)	the name of the axiom if line L is an instance of it
STATES(L,J,P)	states of premise P in the scope of justification set J in line L
STATE(L,[J])	a special state of line L or of its justification set J, respectively

Table 1. Access functions to elements of a proof graph

Selection variables used by all presentation rules

SEL: $L \leftarrow$ <current-line-of-proof>, $C \leftarrow$ CONCLUSION(L),
$J \leftarrow$ ONE-OF($\{L_i | (L_i \in$ JUSTIFICATIONS(L)$) \wedge$ (STATE(L,I) - Summary)$\}$),
$P \leftarrow$ PREMISES(L,J), T $\leftarrow$ MAIN-TERM(L,J)

Definition of the Cut-prop-rule

SEL: $P_1 \leftarrow$ ONE-OF(P), $F_1 \leftarrow$ CONCLUSION(P_1),
$R \leftarrow$ CONCLUSION(T), $F \leftarrow \{$CONCLUSION(P_i)$|P_i \in (P \setminus P_1)\}$
CND: AWARE-OF(User,F_1) $\wedge$ ABLE-INFER(User,R,F,C,F_1)
ACT: STATES(L,J,P_1) $\leftarrow$ STATES(L,J,P_1) $\cup$ Inferable

Definition of the Cut-rule-rule

SEL: $R \leftarrow$ CONCLUSION(T),
$F \leftarrow \{$CONCLUSION(P_i)$|(P_i \in P) \wedge \neg$ (Inferable $\in$ STATES(L,J,P_i))$\}$
CND: AWARE-OF(User,AXIOM(T)) $\wedge$ ABLE-INFER(User,R,F,C,R)
ACT: STATES(L,J,T) $\leftarrow$ STATES(L,J,T) $\cup$ Inferable

Definition of the Compactification-rule

SEL: $F \leftarrow \{$CONCLUSION(P_i)$|(P_i \in P) \wedge \neg$ (Inferable $\in$ STATES(L,J,P_i))$\}$,
$E \leftarrow$ EXPLAINS(L), $E_1 \leftarrow$ ONE-OF(E),
$J_E \leftarrow \{J_i | (J_i \in$ JUSTIFICATIONS(E_1)$) \wedge (L \in$ PREMISES(E_1,I)$)\}$
CND: $\neg$ (STATE(L) = Chainable) $\wedge$ (Inferable $\in$ STATES(L,J,T)) $\wedge$ COHERENT(User,C,F)
ACT: $J_N \leftarrow \cup P_i \in P$:SUBST($L$,$P_i$,$J_E$), JUSTIFICATIONS($E_1$) $\leftarrow$ JUSTIFICATIONS(E_1)$\cup J_N$,
$\forall J_i \in$ JUSTIFICATIONS(E_1),$P_i \in P$: STATES(E_1,J_i,P_i) $\leftarrow$ STATES(E_1,J_i,P_i) $\cup$ Short-cut

Definition of the Restructuring-Rule

SEL: $P_1 \leftarrow$ ONE-OF(P) CND: NO-EXPAND(L) $\vee$ NO-EXPAND(L,J,P_1)
ACT: $N \leftarrow$ MIN($\{I \mid (E_I =$ (EXPLAINS$\circ$ONE-OF)^{I} (L)$) \wedge \neg$ (NO-EXPAND(E_I) $\wedge$ NO- EXPAND
(E_I,ONE-OF(JUSTIFICATIONS(E_I)),P_1))$\}$), $E_1 \leftarrow$ (EXPLAINS$\circ$ONE-OF)^{N} (L),
$J_E \leftarrow \{J_i | (J_i \in$ JUSTIFICATIONS(E_1)$) \wedge (L \in$ PREMISES(E_1,I)$)\}$,
$\forall J_i \in J_E$: PREMISES(E_1,J_i) $\leftarrow$ PREMISES(E_1,J_i) $\cup \{P_1\}$,
STATES(L,J,P_1) $\leftarrow$ STATES(L,J,P_1) $\cup$ Co-reference

Fig. 3. Formalization of the presentation rules

The presentation rules themselves are organized in three components (see Figure 3):

- a *selection set* (SEL), which accesses elements of the current proof line and related ones for consistent access to unique justification sets and premises,
- the *application conditions* (CND), which are evaluated against relevant portions of the proof graph and the presentation knowledge, and
- the *action* (ACT) to be performed if the application conditions are fulfilled.

Line	*Conclusion*	*Main Term*	*Premises*
L_1	$\vdash\ 1 < a$		
L_2	$\vdash\ (0 < 1 \wedge 1 < a) \rightarrow 0 < a$	{{Axiom: Transitivity of <}}	
L_3	$\vdash\ 0 < a \rightarrow 0 \neq a$	{{Axiom: Trichotomy of <}}	
L_4	$\vdash\ 0 \neq a \rightarrow aa^{-1} = 1$	{{Axiom: Inverse Element of K}}	
L_5	$\vdash\ 0 < 1$	{{Lemma}}	
L_6	$\vdash\ 0 < a$	{{L_2 *{Inferable}*	L_1,L_5 *{Inferable}*}}
L_7	$\vdash\ 0 \neq a$	{{L_3 *{Inferable}*, *{L_3 {Inferable},*	L_6}, *L_1 {Short-cut}}*}
L_8 Chainable	$\vdash\ a^{-1} < aa^{-1}$	<Main Term>	<Premises>
L_9 Chainable	$\vdash\ aa^{-1} = 1$	{{L_4 *{Inferable}*, *{L_4 {Inferable},*	L_7 *{Co-reference}*}, *L_1{Co-reference,Short-cut}}*}
L_{10}	$\vdash\ a^{-1} < 1$	{{L_8 *{L_8*	L_9,*L_7*}, *L_9,L_1 {Short-cut}}*}

Fig. 4. A partial proof graph representation, modified by the presentation rules

We explain the generic patterns of these rules in Figure 3, and we demonstrate their effects on the partial proof shown in Figure 4, with modifications printed in italics.

The *Cut-prop*-rule applies to the attainment of a conclusion (C) from a rule (T, main term) and some premises (P). Moreover, the addressee is aware of one of the premises P_1 and is able to infer the necessity of P_1 in order to make that inference (F_1 is the associated formula). Hence, he/she can be assumed to understand the derivation of C without explicit information about P_1. In Figure 4, this applies to premise L_5 $(0 < 1)$ in L_6. If the addressee is aware of $0 < 1$, and is able to conclude $0 < a$ given $1 < a$ and $(0 < 1 \wedge 1 < a) \rightarrow 0 < a$, premise L_5 in the scope of L_6 is marked as *Inferable*.

The *Cut-rule*-rule applies to the attainment of a conclusion (C) from an instantiated rule (T) and some premises (P). If the addressee is aware of that rule (AXIOM(T)) and can infer the instantiated form T from the premises not considered inferable, then he/she can be assumed to understand that inference from these premises alone. In Figure 4, this applies to lines L_6, L_7 and L_9. If the addressee can infer that (1) $0 < a$ in L_6 is concluded from $1 < a$ through transitivity (L_5 being already marked as *Inferable*), (2) $0 \neq a$ is concluded from $0 < a$ in L_7 through trichotomy, and (3) $aa^{-1} = 1$ in L_9 is concluded from $0 \neq a$ through the inverse, then L_2, L_3, and L_4 in the scope of the lines L_6, L_7, and L_9, correspondingly, are marked as *Inferable*.

The *Compactification*-rule applies to two nested and partial justifications (L and P) of a proposition (E_1). If the addressee can infer L from P alone, and the associated formulas, C and F, are coherent, then the addressee can be assumed to understand E_1 directly through P without L. In Figure 4, this applies to a sequence of two inferences. $0 \neq a$ is inferable through $0 < a$ (in L_7), which, in turn, is inferable through $1 < a$ (in L_6). If $0 < a$ and $1 < a$ are considered coherent, $1 < a$ justifies $0 \neq a$ directly.

Finally, the *Restructuring*-rule applies to a proof line L whose premises (or one of them, in case multiple reference motivates restructuring) should not be expanded in

AWARE-OF(User,*T*)::= IN-GENERAL-OR-PROBLEM-SPECIFC-KNOWLEDGE(User,*T*)

ABLE-INFER(User,*M,P,C,U*)::=((($U = M$) $\wedge$ INSTANTIATED(COMPOSE($P \cup C$,*M*))) $\vee$
    (($U \in P$) $\wedge$ INSTANTIATED(COMPOSE(($P \setminus U) \cup C$,*M*)))) $\wedge$
    MATCH(ONE-OF(ABSTRACTED(User,*M*)),GENERIC(*M*))

COHERENT(User,*F*,*G*)::= (($|G| > 1$) $\wedge$ (($\wedge_{G_i \in G} : G_i) \equiv F$)) $\vee$
    (($|G| = 1$) $\wedge$ ($G_1 \in G$) $\wedge$ (SUBFORMULA(F,G_1) $\vee$
    MATCH(ONE-OF(ABSTRACTED(User,*F*)),ONE-OF(ABSTRACTED(USER,G_1)))$\vee$
    (GENERALIZATION(F,G_1) $\wedge$ AWARE-OF(User,GENERALIZATION(F,G_1))) $\vee$
    (INSTANCE(F,G_1) $\wedge$ AWARE-OF(User,INSTANCE(F,G_1)))))

NO-EXPAND(*L*)::= STATE(*L*) = Chainable

NO-EXPAND(*L,J,P*)::= 0 < |EXPLAINS(*P*) \ $\{S \mid S \in$ ((JUSTIFICATIONS$\circ$ONE-OF$\circ$
    PREMISES$\circ$ONE-OF)*(*L*) $\cup$ (JUSTIFICATIONS$\circ$ONE-OF$\circ$MAIN-TERM)*(*L*))$\}$|

Fig. 5. Interpretations of the mental capabilities and presentation preferences

describing L. These premises are marked as *Co-reference* and are copied to the 'nearest' line explained indirectly by L, in which expansion is suitable. In Figure 4, this applies to L_9, marked as *Chainable*, so that each of the alternative premises L_1 ($1 < a$) and L_7 ($0 \neq a$) are marked as *Co-reference* in L_9 and copied to the premises of L_{10}.

So far, we have silently assumed the predicates expressing mental capabilities of the audience and presentation knowledge to evaluate as desired. In the third and last part of this section, we describe domain-specific interpretations of the relevant predicates AWARE-OF, ABLE-INFER, COHERENT, and NO-EXPAND. The interpretations of the three predicates expressing mental capabilities are grounded in two predicates MATCH and INSTANTIATED and two functions COMPOSE and ABSTRACTED, the degree of variation of the latter being adaptable to parameters in a user model:

COMPOSE(*F*,*G*) – substitutes a set of subformulas *F* into the formula *G* from which the elements of *F* were obtained, so that each variable in *G* not covered by one of the subformulas is replaced by a meta-variable.

ABSTRACTED(*User*,*F*) – Formula *F* is abstracted from instantiations made when generating it. Depending on addressee specific parameters, such an abstraction is typically more complex than mere replacements of variables by meta-variables. For example, constants are replaced by a special placeholder, variables are combined with constant factors or certain operators into meta-variables, e.g., $2*a$ or a^{-1} into v rather than $2*v$ or v^{-1}, respectively (v being a meta-variable), or certain operators may be generalized (e.g., '<' and '=' into 'comparison operator'). These criteria are motivated by book proof presentations. Thus, ABSTRACTED may yield several solutions, which are tested one after the other in conditions that contain ABSTRACTED, starting with the simplest one, until a successful match is obtained or all variants fail.

MATCH(*F*,*G*) – an abstracted formula *F* matches with constants and variables in a formula *G* or is identical to another abstracted formula *G* but for namings.

INSTANTIATED(*F*) – all meta-variables in a formula are instantiated.

For assessing the addressee's awareness (AWARE-OF), we test whether a piece of knowledge required is entailed in a list of theorems, definitions, and hierarchical relations assumed to be known to the addressee, which is expressed in a user model as simple stereotypes (see [7]). In Figure 4, this comprises the axioms of transitivity, trichotomy, and inverse, and elementary ordering relations between constants. The underlying simplifying assumption is that being acquainted with some piece of generic knowledge is sufficient to be aware of it in the course of the entire proof.

The inferential capabilities (ABLE-INFER) express whether a user is able to infer an unknown piece of knowledge U, given a valid conclusion C from a main term M and some premises P, without the unknown part U. In logical terms, the addressee must be able to infer $(P,f \vdash C) \Rightarrow (f \rightarrow M)$ if $U = M$, and $(P\backslash U,\mathrm{M},f \vdash C) \Rightarrow (f \rightarrow U)$ if $U \in P$. These inferences are approximated by the requirements that (1) composing the information given is sufficient to fully instantiate the entire inference step, and (2) matching the instantiated and ABSTRACTED rule M with its generic form is within the complexity limitations that the addressee is assumed to be able to handle. $\Rightarrow$

Coherence manifests itself in leaving implicit 'direct' causes in the sense of [22] rather than 'indirect' ones, which require significantly more reasoning effort. In the model of expert system explanations [10], this aspect is assessed by the purposes of arguments. For mathematical proofs, we assess this coherence aspect by comparing the structure of the terms considered, F and G. If the set of terms G consists of more than one element, then a conjunction of its elements must yield F. Otherwise, the single element of G, once ABSTRACTED, must be either equal to F, or it must be a SUBFORMULA, an INSTANTIATION, or a GENERALIZATION of it. In Figure 4, the coherence between $0 \neq a$, $0 < a$ and $1 < a$ holds due to abstraction.

Finally, NO-EXPAND expresses the superiority of presenting an assertion without justifications of its premises in the local proof context. This preference is present for all premises if the proof line L is marked as *Chainable*, that is, it needs to be self-contained or if, for a specific premise, this assertion appears as a justification in at least one proof line which is neither a direct nor an indirect justification of line L.

6 Improvements Obtained in a More Complex Example

In order to illustrate the functionality of our presentation rules in a larger context, we demonstrate how the Steamroller problem [21] is treated by our methods. The problem definition consists of several pieces of 'simplified' real world knowledge, from which the theorem below is to be proven (see the upper part of Figure 6).

In a nutshell, the proof runs along the following lines: Through applying the axiom about the eating habits of animals three times, it is first derived that birds eat plants, then that foxes do not eat grains and, finally, that foxes eat the smaller grain-eating birds, the last being the example needed to prove the theorem in this problem.

The proof graph obtained by applying the theorem prover OTTER [19] and transforming the result to the assertion level consists of 51 nodes. Without the new presentation techniques, PROVERB produces a full page of text from this proof graph, which we believe is pretty much comparable in length to what other systems would do. This text contains many trivially inferable reasoning steps, mostly categorial ones, generalizations, and instantiations. In addition, the assertion level, which supports generating concise and more natural texts, leads to a particular problem here, since it only provides condensed descriptions of the three key inference steps.

Problem definition:

(1) Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them. Also there are some grains, and grains are plants.

(2) Every animal either likes to eat all plants (2a) or all animals smaller than itself that like to eat some plants (2b).

(3) Caterpillars and snails are smaller than birds, which are smaller than foxes, which are smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars, but not snails. Caterpillars and snails like to eat plants.

(4) Therefore there is an animal that likes to eat a grain-eating animal. (*Theorem*)

$$\underline{\mathrm{EATS}(w,g) \vee ((\mathrm{EATS}(f,g) \wedge (f < w))} \Rightarrow \underline{\mathrm{EATS}(w,f))}\ _{\text{Assertion}},\ \neg\mathrm{EATS}(w,g),\ \neg\mathrm{EATS}(w,f),\ f < w$$
$$\neg\mathrm{EATS}(f,g) \qquad \text{(Assertion level)}$$

$$\underline{\mathrm{EATS}(w,g) \vee ((\mathrm{EATS}(f,g) \wedge\ (f < w)) \Rightarrow \mathrm{EATS}(w,f))}\ _{\vee\mathrm{E}},\ \neg\mathrm{EATS}(w,g)$$
$$\underline{(\mathrm{EATS}(f,g) \wedge\ (f < w)) \Rightarrow \mathrm{EATS}(w,f)}\ _{\text{Modus tollens}},\ \neg\mathrm{EATS}(w,f)$$
$$\underline{\neg\mathrm{EATS}(f,g) \vee \neg\ (f < w)}\ _{\vee\mathrm{E}},\ f < w \qquad \text{(Partial assertion level)}$$
$$\neg\mathrm{EATS}(f,g)$$

Line		*Conclusion*	*Main Term*	*Premises*
L_1	⊢	$\forall x,y: ((\mathrm{SNAIL}(x) \wedge \mathrm{PLANT}(y)) \Rightarrow \mathrm{EATS}(x,y))$	{{Lemma}}	
L_2	⊢	$\mathrm{SNAIL}(s)$	{{Hypothesis}}	
L_3	⊢	$\mathrm{GRAIN}(g)$	{{Hypothesis}}	
L_4	⊢	$\forall x: (\mathrm{GRAIN}(x) \Rightarrow \mathrm{PLANT}(x))$	{{Lemma *{Inferable}*}}	
L_5	⊢	$\mathrm{GRAIN}(g) \Rightarrow \mathrm{PLANT}(g)$	{{L_4 *{Inferable}* }}	
L_6	⊢	$\mathrm{PLANT}(g)$	{{L_5 *{Inferable}*	L_3}}
L_7	⊢	$(\mathrm{SNAIL}(s) \wedge \mathrm{PLANT}(g)) \Rightarrow \mathrm{EATS}(s,g)$		{{L_1 *{Inferable}*}}
L_8	⊢	$\mathrm{SNAIL}(s) \wedge \mathrm{PLANT}(g)$	{{L_2	L_6},
			{L_2	*L_3 {Short-cut}}* }
L_9	⊢	$\mathrm{EATS}(s,g)$	{{L_7	L_8},
			{L_7	*L_2 {Short-cut}, L_6 {Short-cut}},*
			{L_7	*L_2 {Short-cut}, L_3 {Short-cut}},*
			{L_1 {Inferable,Short-cut}	*L_8},*
			{L_1 {Inferable,Short-cut}	*L_2 {Short-cut}, L_6 {Short-cut}},*
			{L_1 {Inferable,Short-cut}	*L_2 {Short-cut}, L_3 {Short-cut}}* }

Proof:

(5) Let *s* be a snail, *b* a bird, *f* a fox, *g* a grain, and *w* a wolf. Since *s* is smaller than *b*, s eats *g*, and *b* does not eat *s*, (2b) does not hold for *b*. Hence, *b* eats *g*. Since *w* does not eat *g*, (2b) holds for *w*. Since *w* does not eat *f*, either *f* does not eat *g* or *f* is not smaller than *w*. Since *f* is smaller than *w*, *f* does not eat *g*. Hence, (2b) holds for *f*. Since *b* is smaller than *f* and *b* eats *g*, *f* eats *b*. Hence (4).

Fig. 6. The Steamroller problem – specification, proof details, and presentation

This presentation can be improved significantly by our methods, provided the audience is credited with knowing the categories of the relevant animals and plants.

The involved applications of the key lemma (labeled (2) in Figure 6) are expanded to the partial assertion level, and justifications inferable on the basis of knowledge attributed to the audience as well as short-cuts are incorporated into the proof graph by adding the appropriate marks. Restructuring applies only to assumptions about instantiated animals or plants, which are put in front due to domain conventions.

Expanding the involved assertion level step proving that *f*, a fox, does not eat *g*, a plant, is shown in Figure 6, second section from the top. The other two involved assertion level steps are also applications of the key axiom, and they are expanded in a similar way. The section below shows a partial trace justifying EATS(*s*,*g*), which contributes to these partial assertion level inferences. This trace demonstrates that EATS(*s*,*g*) is considered inferable from the assumptions SNAIL(*s*) and GRAIN(*g*), on the basis of the assumed background knowledge. Similarly, all premises of partial assertion level steps are either inferable from background knowledge or from single proof assumptions introduced explicitly in the first part of the presentation.

The text at the bottom of Figure 6 illustrates the improvements obtained. It is, however, polished in a crucial aspect, the textual reference to partial axioms through labels ((2a) and (2b)), and their partial instantiations (e.g., '(2a) for *f*''), which only works for complete formulas in the current version of PROVERB. The text is still stereotypical and lacks structure signalling hints, compared to good human presentations. Nevertheless, it is more concise than previous versions, though involved steps are explained in more detail, which also makes the text better comprehensible.

7 Discussion

In this section, we discuss the choice of the assertion level as basic representation for our model, the assumptions incorporated, the computational effort involved in some parts, and extensions in functionality and for overcoming simplifying assumptions.

It is our firm belief that the assertion level is the most suitable basic representation level for presenting proofs, because lemma application, its level of granularity, is the best approximation to the flexible, knowledge-intensive human reasoning on the level of a uniform logical calculus. It is sometimes too coarse-grained, in case of involved lemma applications, and it is too detailed in situations where extra knowledge in human reasoning compensates for nested applications of axioms. The expansion to the partial assertion level and the application of the presentation rules address these two issues. In purely logical terms, the role of a presentation rule is to replace parts of the justifications, by enhancing the strictly logical calculus through contextually motivated background knowledge.

Not unexpectedly, transforming proofs to the assertion level requires considerable computational effort for larger proofs, but it is still bearable, as the procedure that transforms a refutation graph into a proof at the assertional level shows [18]. It typically requires slightly more than one minute for large proofs, such as that of the Robins problem [20] and, unlike the original procedure developed by Huang [14], it is not restricted to unit resolution. A similar computational effort is required for expansion to the ND-level for proofs of comparable size. This expansion and partial recomposition can hardly be avoided, because the uniform refutation graph representation does not enable the identification of syllogistic reasoning patterns. Overall, applying the presentation rules is computationally less costly, since the effort roughly correlates to the proof tree size by a constant factor, which may be increased by multiple used justifications and alternative paths through short-cuts. The only potentially expensive operation is the function ABSTRACTED, depending on the complexity of axioms considered and the variability of the operations permitted in the abstraction step. Altogether, these considerations make it clear that a good presentation always requires a certain computational effort, but fully elaborate presentations in a single shot do only make sense for proofs up to a certain complexity.

The current formalization of the presentation rules contains some simplifying assumptions which may turn out to be problematic in larger or less standardized proofs. Awareness of an axiom at any phase of a proof merely requires knowing that axiom, which seems to be sufficient for commonly used axioms only (e.g., transitivity). Moreover, the relevance of a rule for the predicate ABLE-INFER merely requires its full instantiation on the basis of the information given, but neglects eventual ambiguities with other candidate axioms and alternative ways of instantiation. Though some of them may be resolved through communicative prominence of one interpretation over another, several cases may easily turn out to be confusing. Finally, the predicate AWARE-OF relies on a fully elaborate set of assumptions about the audience's mental capabilities that are relevant to a proof considered, which must be declared as parts of the problem specification, similar to new lexical knowledge.

In the future, we intend to extend our model in several ways. We intend to incorporate finer-grained distinctions for awareness and abstraction capabilities, based on classifications of theorems and the proof structure, as well as preferences to handle ambiguities, along the lines of the presentation rules developed for expert system explanations [10], which guide the selection among potentially applicable candidate rules and alternative sets of individuals to which they apply. Moreover, a connection of assumptions about the audience to the database of mathematical theorems in Ω exploits generalizations across problems, thereby greatly reducing the specification problem. Finally, keeping track of assumptions made in applying presentation rules can be exploited for other communicative purposes, such as summaries, presentations for users with divergent knowledge, and interactive presentations.

8 Conclusion

In this paper, we have described a new method for presenting machine-found proofs in a human-oriented way. The method relies on a proof represented at the assertion level as introduced by Huang. This representation is modified through expanding involved theorem applications and omitting intermediate reasoning steps, motivated by assumed knowledge and inferential capabilities of the audience. A key part of that model are presentation rules, which express communicatively motivated inferences humans typically make, with special interpretations of the concepts awareness, inferential ability, and coherence, reflecting particularities of mathematical proofs. Though some of the assumptions and heuristics involved are simplified to a certain extent, our method works significantly better than previous ones for non-trivial proofs, and presentations resemble closer those found in text books. In addition, our approach has the potential for a variety of extended functionalities, including alternative presentation purposes, such as summaries, participating in interactive proof presentations, and giving alternative presentations to divergent kinds of addressees.

References

1. Christoph Benzmüller, Lassaad Cheikhrouhou, Detlef Fehrer, Armin Fiedler, Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, Wolf Schaarschmidt, Järg Siekmann, Volker Sorge. …MEGA: Towards a Mathematical Assistant. In *Proc. of CADE-97*, 1997.
2. Dan Chester. The Translation of Formal Proofs into English. In *Artificial Intelligence* 7, pp. 261-278, 1976.
3. Yann Coscoy, Gilles Kahn, Laurent Th ry, Extracting Text from Proof. In *Typed Lambda Calculus and its Application*, 1995.
4. Ingo Dahn. Using ILF as a User Interface for Many Theorem Provers. In *Proc. of User Interfaces for Theorem Provers*, Eindhoven, 1998.
5. A. Edgar, Francis Pelletier. Natural Language Explanation of Natural Deduction Proofs. In *Proc. of the First Conference of the Pacific Association for Computational Linguistics*, Simon Fraser University, 1993.
6. Detlef Fehrer, Helmut Horacek. Presenting Inequations in Mathematical Proofs, to appear in *Information Sciences*, Special Issue on Logical Methods for Computational Intelligence, 1999.
7. Detlef Fehrer, Helmut Horacek. Exploiting the Addressee's Inferential Capabilities in Presenting Mathematical Proofs, In *Proc. of IJCAI-97*, pp. 556-560, Nagoya, 1997.
8. Nancy Green, Sandra Carberry. A Hybrid Reasoning Model for Indirect Answers. In *Proc. of ACL-94*, Las Cruces, New Mexico, 1994.
9. H. Grice. Logic and Conversation. In *Syntax and Semantics: Vol. 3, Speech Acts*, pp. 43-58, Academic Press, 1975.
10. Helmut Horacek. A Model for Adapting Explanations to the User's Likely Inferences. *User Modeling and User Adapted Interaction,* 7, pp. 1-55, 1997.
11. Helmut Horacek. Generating Inference-Rich Discourse Through Revisions of RST-Trees. In *Proc. of AAAI-98,* pp. 814-820, 1998.
12. Xiaorong Huang. Human Oriented Proof Presentation: A Reconstructive Approach. PhD Dissertation, University of the Saarland, 1994.
13. Xiaorong Huang. Reconstructing Proofs at the Assertional Level. In *Proc. of CADE-94*, pp. 738-752, 1994.
14. Xiaorong Huang. Translating Machine-Generated Proofs into ND-Proofs at the Assertion Level. In *Proc. of PRICAI-96*, LNAI, Springer, 1996.
15. Xiaorong Huang, Armin Fiedler. Proof Presentation as an Application of NLG. In *Proc. of IJCAI-97*, pp. 965-971, Nagoya, Japan, 1997.
16. Philip Johnson-Laird, Ruth Byrne. Deduction. Ablex Publishing, 1990.
17. William Mann, Sandra Thompson. Rhetorical Structure Theory: A Theory of Text Organization. Technical Report, ISI/RR-83-115, ISI at University of Southern California, 1983.
18. Andreas Meier. †bersetzung automatisch erzeugter Beweise auf Faktenebene. Diploma thesis, University of the Saarland, 1997.
19. William McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, 1994.
20. William McCune. Solution of the Robins Problem. *Journal of Automated Reasoning* 19(3), pp. 263-276, 1997.
21. Mark Stickel. Schubert's Steamroller Problem: Formulations and Solutions. In *Journal of Automated Reasoning* 2(1), 1986.
22. Manfred Thüring, Kurt Wender. †ber kausale Inferenzen beim Lesen. In *Sprache und Kognition* 2, pp. 76-86, 1985.
23. Marilyn Walker. The Effect of Resource Limits and Task Complexity on Collaborative Planning in Dialogue. In *Artificial Intelligence* 85, pp. 181-243, 1996.
24. Ingrid Zukerman, Richard McConachy. Generating Concise Discourse that Addresses a User's Inferences. In *Proc. of IJCAI-93*, pp. 1202-1207, Chambery, France, 1993.

On the Universal Theory of Varieties of Distributive Lattices with Operators: Some Decidability and Complexity Results

Viorica Sofronie-Stokkermans

Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany
sofronie@mpi-sb.mpg.de

Abstract. In this paper we establish a link between satisfiability of universal sentences with respect to varieties of distributive lattices with operators and satisfiability with respect to certain classes of relational structures. We use these results for giving a method for translation to clause form of universal sentences in such varieties, and then use results from automated theorem proving to obtain decidability and complexity results for the universal theory of some such varieties.

1 Introduction

In this paper we give a method for automated theorem proving in the universal theory of certain varieties of distributive lattices with well-behaved operators. For this purpose, we use extensions of Priestley's representation theorem for distributive lattices. The advantage of our method is that we avoid the explicit use of the full algebraic structure of such lattices, instead using sets endowed with a reflexive and transitive relation and with additional functions and relations that correspond to the operators in the lattices in a standard way. Our interest in such algebras is motivated by the fact that many existing non-classical logics are sound and complete with respect to varieties of distributive lattices with additional well-behaved operators. Moreover, uniform word problems in lattices also occur in more general contexts such as database dependency theory [6].

The main contributions of this paper are the following:

- We establish a link between satisfiability of universal sentences with respect to varieties of distributive lattices with operators and satisfiability with respect to classes of relational structures. This extends the results from [19].
- We use these results for giving a method for translation to clause form of universal sentences in such varieties.
- We use existing results from automated theorem proving to obtain decidability and complexity results.

We first studied this type of relationships in the context of finitely-valued logics in [18], and then extended the ideas to certain classes of non-classical logics in [20]. This paper shows that the idea is much more general, and can be used for the whole universal theory of certain varieties of distributive lattices with operators. In particular, the method presented here subsumes in a natural way both existing methods for translating modal logics to classical logic and methods for automated theorem proving in finitely-valued logics based on distributive lattices with operators. The approach has the following advantages:

– It avoids the problems that occur when ACI-operators have to be considered (as is the case in algebraic automated reasoning for lattices).
– Known saturation-based techniques for theories of reflexive and transitive relations, such as ordered chaining with selection, can be used successfully.
– Decidability and complexity results follow in many cases as consequences of existing decision procedures based on ordered resolution or ordered chaining.
– We obtain decidability and complexity results for uniform word problems in certain non locally finite varieties of distributive algebras with operators (as far as we know, no such results were known).
– Considerations concerning the structure of the sets of clauses generated with our method make certain algebraic properties of these varieties visible.

The applicability of our method depends on the possibility of finding the appropriate relational structures that can replace the algebras in the variety in the automated theorem proving process. It is known from modal logic that such structures may not always exist. Another limitation is given by the fact that, in general, resolution is a semi-decision procedure, and it may be hard or impossible to obtain resolution-based decision procedures for the classes of clauses generated by the method we describe. However, we show that in many cases the method is applicable and leads to decision procedures.

The idea of using representation theorems for establishing a link between the algebraic and relational semantics of non-classical logics goes back to Jónsson and Tarski [11], who for this purpose used an extension of Stone's representation theorem for Boolean algebras with operators. Our work is influenced by the results of Goldblatt [9], who showed that the "modal case" is an illustration of more general results from universal algebra. He gives an extension of the Priestley duality to join and meet hemimorphisms, which we extended in [19] to lattices endowed with certain classes of anti(hemi)morphisms. In this paper we use the results in [9] and [19] and show that the use of representation theorems has applications which range far beyond the area of applications in modal logics.

The paper is structured as follows. In Section 2 the main notions and results needed in the paper are presented. Section 3 contains the main results. Section 4 contains some run examples and a comparison to a more standard approach. Section 5 contains some conclusions and plans for future work.

2 Preliminaries

This section contains the main notions and results needed in this paper.

Partially Ordered Sets and Lattices. We assume known standard notions, such as partially-ordered set, order-filter and order-ideal in a partially-ordered set, cf. [7]. Given a partially-ordered set $(X, \leq)$, by $\mathcal{O}(X)$ we denote the set of order-filters of X. A *lattice* is a partially-ordered set $(L, \leq)$ with the property that every two elements $x, y \in L$ have a supremum and an infimum (denoted $x \vee y$ resp. $x \wedge y$) in L. Alternatively, a non-empty set L together with two binary operations $\vee$ and $\wedge$ on L is called *lattice* if $\vee$ and $\wedge$ are associative, commutative and idempotent and satisfy the absorption laws. A *distributive lattice* is a lattice that satisfies either of the distributive laws. A lattice L has a *first element* if there is an element $0 \in L$ such that $0 \leq x$ for every $x \in L$; it has a *last element* if there is an element $1 \in L$ such that $x \leq 1$ for every $x \in L$. A lattice having both a first and a last element is called *bounded.* The *pseudocomplement* of an element $a \in L$ (denoted by $\neg a$) is the largest element of $\{c \in L \mid a \wedge c = 0\}$ (if any). Given $a, b \in L$, the *pseudocomplement of* a *relative to* b (denoted by $a \Rightarrow b$) is the largest element of $\{c \in L \mid a \wedge c \leq b\}$ (if any). A filter in a lattice L is a non-empty order-filter closed under meets. A filter F is said to be *prime* if $F \neq L$ and for every $x, y \in L$, if $x \vee y \in F$ then $x \in F$ or $y \in F$. Ideals and prime ideals are defined dually.

Priestley Representation for Bounded Distributive Lattices. The Priestley representation theorem [16] states that every bounded distributive lattice A is isomorphic to the lattice of clopen (i.e. closed and open) order filters of the ordered topological space having as points the prime filters of A, ordered by inclusion, and the topology generated by the sets of the form $X_a = \{F \mid F \text{ prime filter}, a \in F\}$ and their complements as a subbasis. The partially ordered set of all prime filters of A, ordered by inclusion, and endowed with the topology mentioned above will be denoted $D(A)$ (we will refer to it as the dual of A). If we denote the lattice of clopen order filters of an ordered topological space X by $\mathsf{ClopenOF}(X)$, the Priestley representation theorem states that there exists an isomorphism of bounded lattices, $\eta_A : A \to \mathsf{ClopenOF}(D(A))$.

Universal Algebra. For the necessary notions of universal algebra we refer e.g. to [5]. For every signature Σ and every arity function $a : \Sigma \to \mathbb{N}$, a Σ-algebra is a structure $(A, \{\sigma_A\}_{\sigma \in \Sigma})$, where for every $\sigma \in \Sigma$, $\sigma_A : A^{a(\sigma)} \to A$. If the signature Σ is known we may use the notation A for the Σ-algebra $(A, \{\sigma_A\}_{\sigma \in \Sigma})$. A Σ-algebra A has a *bounded distributive lattice reduct* if there exist operations $\vee, \wedge, 0, 1$ in Σ such that $(A, 0, 1, \vee_A, \wedge_A)$ is a bounded distributive lattice. A *distributive p-lattice* (resp. *Heyting algebra*) is an algebra $(A, 0, 1, \vee, \wedge, \neg)$ (resp. $(A, 0, 1, \vee, \wedge, \Rightarrow, \neg)$) with a bounded distributive lattice reduct such that for every $a, b \in A$, $\neg a$ is the pseudocomplement of a, and $a \Rightarrow b$ is the relative pseudocomplement of a with respect to b.

Given a set X, the term algebra over Σ in the variables X will be denoted $\mathsf{Term}_\Sigma(X)$. An *equation* is an expression of the form $t_1 = t_2$ where $t_1, t_2 \in \mathsf{Term}_\Sigma(X)$; an *implication* is an expression of the form $\beta_1 \wedge \cdots \wedge \beta_m \to \alpha$, where $\beta_1, \ldots, \beta_m, \alpha$ are equations. A *conditional equation* (or *quasi-equation*) is an expression which is either an equation or an implication. A Σ-algebra A *satisfies* a quasi-equation γ (notation: $A \models \gamma$) if the quasi-equation is true for every substitution of elements in A for the variables. A class $\mathcal{K}$ of algebras satisfies γ (notation: $\mathcal{K} \models \gamma$) iff all algebras in $\mathcal{K}$ satisfy γ. Truth of conditional equations is preserved under isomorphic images, subalgebras, and products. Truth of equations is additionally preserved under homomorphic images. A variety is the class of all algebras that satisfy a set of identities, or, alternatively, a class of algebras which is closed under homomorphic images, subalgebras and direct products.

Logic. Let $\mathcal{K}$ be a class of algebras. The *elementary theory* of $\mathcal{K}$ is the collection of all closed formulae in first-order predicate logic with equality that are valid in $\mathcal{K}$. The *universal theory* of $\mathcal{K}$ is the collection of those closed formulae valid in $\mathcal{K}$ which are of the form $\forall x_1 \dots \forall x_k (\bigwedge_{i=1}^{m} ((\neg) t_{i1} = s_{i1} \vee \dots \vee (\neg) t_{in_i} = s_{in_i}))$. The *universal Horn theory* of $\mathcal{K}$ is the collection of those closed formulae valid in $\mathcal{K}$ which are of the form $\forall x_1 \dots \forall x_k (t_{11} = t_{12} \wedge \dots \wedge t_{n1} = t_{n2} \rightarrow s_1 = s_2)$. The *equational theory* of $\mathcal{K}$ is the set of all closed formulae valid in $\mathcal{K}$ which are of the form $\forall x_1 \dots x_k (t = s)$. Given a recursively enumerable set E of conditional Σ-equations we say that the *word problem* for E is decidable if we can decide for every $t, s \in \mathsf{Term}_\Sigma(X)$ whether $s \equiv_E t$, where $\equiv_E$ denotes the congruence on $\mathsf{Term}_\Sigma(X)$ generated by E. We say that the *uniform word problem* for E is decidable if the universal Horn theory of the class of all models of E is decidable. McKinsey [13] showed that for every class $\mathcal{K}$ of Σ-algebras which is closed under direct products, if a sentence of the form

$$\forall x_1 \dots \forall x_k (s_{11} = s_{12} \wedge \dots \wedge s_{n1} = s_{n2} \rightarrow t_{11} = t_{12} \vee \dots \vee t_{m1} = t_{m2})$$

is true in $\mathcal{K}$, then there exists $j \in \{1, \dots, m\}$ such that

$$\forall x_1 \dots \forall x_k (s_{11} = s_{12} \wedge \dots \wedge s_{n1} = s_{n2} \rightarrow t_{j1} = t_{j2})$$

is true in $\mathcal{K}$. In particular it follows that for every class $\mathcal{K}$ of algebras which is closed under direct products, if its universal Horn theory is decidable, then its universal theory is decidable.

Decidability Results for Distributive Lattices. Decidability of the theories related to various classes of algebras has been studied extensively. In what follows we will present existing decidability and complexity results for the variety of distributive lattices. It is known (cf. e.g. [4], p.16) that the elementary theory of every non-trivial variety of lattices is undecidable. Thus, the elementary theory of the variety DLat of distributive lattices is undecidable. The uniform word problem for distributive lattices is decidable (since $\mathsf{DLat} = ISP(2)$, where 2 is the 2-element lattice), and has been proved to be co-NP-hard by Bloniarz et al. [10]. By the result of McKinsey [13] mentioned above it follows that the universal theory of the variety of distributive lattices is decidable. (In 1920, Skolem [17] gave a polynomial time decision procedure for the uniform word problem for general lattices, which cannot be used for the variety of distributive lattices.)

Struth [21] gives a calculus based on non-symmetric rewriting (modulo ACI) for the elementary theory of finite distributive lattices. Besides the possibility of extending this calculus to families of well-behaved operators on lattices, and the complexity results established for (boolean) Tarskian set constraints by McAllester et al. [12], and Mielniczuk and Pacholski [14], we are not aware of any systematic study on automated theorem proving or decidability and complexity results for varieties of distributive lattices with additional operators.

Resolution as a Decision Procedure. We assume known the usual notions and notations in first-order logic and resolution. For details we refer to any text on automated theorem proving. Unrefined resolution is only a semi-decision procedure for first-order logic. However, for some classes of formulae known to be decidable, the resolution principle can be adapted in order to obtain decision procedures. The main idea is to find a complete resolution refinement (usually an ordering refinement, possibly combined with the use of a selection function) which is terminating on the specified class of clauses. Termination may be proved for instance by finding a depth and a length limit for the resolvents.

In this paper reflexive and transitive relations will play an important rôle. In the presence of this kind of relations, superposition and ordered chaining have successfully been used for obtaining decidability results. The *superposition calculus* is a refutationally complete inference system for arbitrary first-order clauses with equality. Its inference rules are restricted versions of paramodulation, resolution, and factoring, parametrized by a total reduction ordering $\succ$ on ground expressions and a selection function S. The *ordered chaining calculus* is an extension of the superposition calculus to more general reflexive and transitive relations. Its inference rules are restricted versions of (positive and negative) chaining, resolution, and factoring, parametrized by a total reduction ordering $\succ$ on ground expressions and a selection function S. In both cases, S assigns to each clause a (possibly empty) multiset of negative literals. For details cf. [1,2]. Superposition with selection and simplification has been proved to be a decision procedure for the monadic class with equality [3]. Ordered chaining with selection was used to obtain decision procedures for the relational translation of propositional modal logics with modal operators satisfying the axiom 4 [8].

3 On the Universal Theory of Subvarieties of DLO_Σ

We start by presenting some results on a Priestley representation for distributive lattices with operators. We show that this helps to establish a link between satisfiability of universal sentences with respect to varieties of distributive lattices with operators and satisfiability with respect to certain classes of relational structures. These results are used for giving a method for translation to clause form of universal sentences in such varieties.

Definition 1. *Let A be an algebra with a bounded lattice reduct. A* lattice antimorphism *on A is a function $k : A \to A$ which maps 0 to 1, 1 to 0, joins to meets and meets to joins. A* join hemimorphism *on A is a function $f : A^n \to A$ that preserves 0 and all finite joins in every argument. A* meet hemimorphism *on A is a function $g : A^n \to A$ that preserves 1 and all finite meets in every argument. A* join hemiantimorphism *on A is a function $f' : A^n \to A$ that maps 1 to 0 and meets to joins in every argument. A* meet hemiantimorphism *on A is a function $g' : A^n \to A$ that maps 0 to 1 and joins to meets in every argument.*

Let Σ be a signature containing function symbols in several classes; in order to distinguish these classes, we will write $\Sigma = Lh \cup La \cup Jh \cup Mh \cup Ja \cup Ma$, where

Lh, La, Jh, Mh, Ja, and Ma may be empty. Let DLO_Σ be the class of all bounded distributive lattices with operators in Σ, $(A, \vee, \wedge, 0, 1, \{\sigma_A\}_{\sigma \in \Sigma})$, such that if σ is an operation symbol in Lh, La, Jh, Mh, Ja, or Ma, then σ_A is, respectively, a lattice homomorphism, lattice antimorphism, join or meet hemimorphism, or join or meet hemiantimorphism. DLO_Σ is a variety.

3.1 Priestley Representation for DLO_Σ and Σ-Relational Structures

In [19] we showed that, given an algebra $A \in \mathsf{DLO}_\Sigma$, the operators in Σ induce in a canonical way functions and relations on its Priestley dual $D(A)$ which, in their turn, induce operators on $\mathsf{ClopenOF}(D(A))$. Taking into account these correspondences, we showed that the canonical isomorphism $\eta_A : A \to \mathsf{ClopenOF}(D(A))$ from the Priestley duality is an isomorphism of algebras in DLO_Σ. For details, including a categorical duality theorem, we refer to [7,9,18,19]. The Priestley duality has been extended to distributive p-lattices and Heyting algebras (cf. e.g. [15], [9]). The dual spaces $(X, \leq, \tau)$ satisfy in this case the additional condition that for every clopen order-filter U, $X \backslash \downarrow U$ is clopen.

Definition 2. *Let $(X, \leq)$ be such that $\leq$ is a reflexive and transitive relation on X, and let $R \subseteq X^{n+1}$. R is called* increasing *if for every $\overline{x} \in X^n$ and every $y, z \in X$, if $R(\overline{x}, y)$ and $y \leq z$ then $R(\overline{x}, z)$; R is called* decreasing *if for every $\overline{x} \in X^n$ and every $y, z \in X$, if $R(\overline{x}, y)$ and $z \leq y$ then $R(\overline{x}, z)$.*

For every set X endowed with a reflexive and transitive relation $\leq$, its set $\mathcal{H}(X)$ of hereditary (i.e. upwards-closed with respect to $\leq$) subsets can be endowed with a bounded lattice structure (where join is union, meet is intersection, $0 = \emptyset$ and $1 = X$). We can canonically define additional operators on $\mathcal{H}(X)$ as showed below.

Theorem 1. *Let $(X, \leq)$ be a set endowed with a reflexive and transitive relation.*

(1) Every $\leq$-preserving map $H_X : X \to X$ induces a lattice morphism $h_H : \mathcal{H}(X) \to \mathcal{H}(X)$, defined for every $U \in \mathcal{H}(X)$ by $h_H(U) = H_X^{-1}(U)$.
(2) Every $\leq$-reversing map $K_X : X \to X$ induces a lattice antimorphism $k_K : \mathcal{H}(X) \to \mathcal{H}(X)$, defined for every $U \in \mathcal{H}(X)$ by $k_K(U) = X \backslash K_X^{-1}(U)$.
(3) Every increasing relation $R_X \subseteq X^{n+1}$ induces a join hemimorphism $f_R : \mathcal{H}(X)^n \to \mathcal{H}(X)$, and a join hemiantimorphism $f'_R : \mathcal{H}(X)^n \to \mathcal{H}(X)$, defined for every $U_1, \dots, U_n \in \mathcal{H}(X)$ by:
$f_R(U_1, \dots, U_n) = \{x \in X \mid \exists x_1, \dots, x_n(x_i \in U_i \textit{ for all } i, \textit{ and } R_X(x_1, \dots, x_n, x))\}$,
$f'_R(U_1, \dots, U_n) = \{x \in X \mid \exists x_1, \dots, x_n(x_i \notin U_i \textit{ for all } i, \textit{ and } R_X(x_1, \dots, x_n, x))\}$.
(4) Every decreasing relation $Q_X \subseteq X^{n+1}$ induces a meet hemimorphism $g_Q : \mathcal{H}(X)^n \to \mathcal{H}(X)$, and a meet hemiantimorphism $g'_Q : \mathcal{H}(X)^n \to \mathcal{H}(X)$, defined for every $U_1, \dots, U_n \in \mathcal{H}(X)$ by:
$g_Q(U_1, \dots, U_n) = \{x \in X \mid \forall x_1, \dots, x_n \, (Q_X(x_1, \dots, x_n, x) \to \exists i, x_i \in U_i)\}$,
$g'_Q(U_1, \dots, U_n) = \{x \in X \mid \forall x_1, \dots, x_n \, (Q_X(x_1, \dots, x_n, x) \to \exists i, x_i \notin U_i)\}$.
(5) Moreover, a pseudocomplementation $\neg$ and a relative pseudocomplementation $\Rightarrow$ can be defined on $\mathcal{H}(X)$ by $\neg U = \{x \mid \forall y(x \leq y \to y \notin V)\}$ and $U \Rightarrow V = \{x \mid \forall y((x \leq y \wedge y \in U) \to y \in V)\}$.

Proof: *(Sketch)* The proof closely follows the proof of the similar results established in [9,19] for relational structures endowed with partial orders. It can be seen that the antisymmetry of $\leq$ is not needed anywhere in the proof. □

Let $\Sigma = Lh \cup La \cup Jh \cup Mh \cup Ja \cup Ma$ be a signature as discussed above.

Definition 3. *An RT Σ-relational structure is a set endowed with a reflexive and transitive relation $\leq$ and with additional maps and relations indexed by Σ, $(X, \leq, \{\sigma_X\}_{\sigma \in \Sigma})$, where if $\sigma \in Lh$, $\sigma_X : X \to X$ is a $\leq$-preserving map, if $\sigma \in La$, $\sigma_X : X \to X$ is a $\leq$-reversing map, if $\sigma \in Jh \cup Ja$ with arity n, $\sigma_X \subseteq X^{n+1}$ is an increasing relation, and if $\sigma \in Mh \cup Ma$ with arity n, $\sigma_X \subseteq X^{n+1}$ is a decreasing relation.*

The class of *RT* Σ-relational structures will be denoted by RTS_Σ. For every $X \in RTS_\Sigma$ and every $\sigma \in \Sigma$ let $\sigma_{\mathcal{H}(X)}$ be the operation on $\mathcal{H}(X)$ associated with σ_X as explained in Theorem 1. The corresponding algebra is again denoted by $\mathcal{H}(X)$. By Theorem 1, $\mathcal{H}(X) \in \mathsf{DLO}_\Sigma$. Conversely, for every $A \in \mathsf{DLO}_\Sigma$, the ordered space $U(D(A))$, obtained from $D(A)$ by ignoring the topology, is in RTS_Σ. $\mathsf{ClopenOF}(D(A))$ is a subalgebra (in DLO_Σ) of $\mathcal{H}(D(A)) = \mathcal{O}(D(A))$.

Notation. As a convention, if not explicitly specified otherwise, in what follows h (resp. k) will denote an operation symbol in Lh (resp. La), f one in $Jh \cup Ja$, and g one in $Mh \cup Ma$. Sometimes, in order to distinguish between elements in Jh and Ja, resp. Mh and Ma, the operation symbols in Ja and Ma will be denoted by f' resp. g'. The symbols in $Jh \cup \cdots \cup Ma$ are interpreted as maps for elements in DLO_Σ, and as relations in RTS_Σ. For the sake of clarity we will always overline the operation symbol in the latter case. In particular, in Section 3.3 (Theorem 3) and Section 3.4 the function resp. relation symbols h, k, f, g are in the classes corresponding to the labeling in (Ren).

Let $\phi = \forall x_1, \ldots, x_k (\bigwedge_{i=1}^n s_{i1} = s_{i2} \to \bigvee_{j=1}^m t_{j1} = t_{j2})$ (where $s_{il}, t_{jp} \in \mathsf{Term}_{\Sigma'}(\{x_1, \ldots, x_k\})$, and Σ' is $\Sigma \cup \{\vee, \wedge, 0, 1\}$ to which possibly $\neg$ and $\Rightarrow$ are adjoined). $ST(\phi)$ denotes the set of all subterms of s_{il} and t_{jp}, $1 \leq i \leq n, 1 \leq j \leq m, l, p \in \{1, 2\}$, $ns = |ST(\phi)|$, $nf = |Lh \cup La|$, $np = |Jh \cup Ja \cup Mh \cup Ma|$, and mp is the maximal arity of an operation in $Jh \cup Ja \cup Mh \cup Ma$.

3.2 A Link between Algebraic and Relational Models

We study the link between satisfiability of universal sentences with respect to algebraic and relational models. As algebraic models we consider subvarieties $\mathcal{V}$ of DLO_Σ (possibly with an additional p-lattice or Heyting algebra structure), satisfying the condition (K) below:

(K) There exists a class $\mathcal{K}$ of *RT* Σ-relational structures such that:
 (i) for every $A \in \mathcal{V}$, the *RT* Σ-relational structure $U(D(A))$ is in $\mathcal{K}$;
 (ii) for every $X \in \mathcal{K}$, the algebra $\mathcal{H}(X)$ is in $\mathcal{V}$.

Theorem 2. *Let $\phi = \forall x_1, \ldots, x_k (\bigwedge_{i=1}^n s_{i1} = s_{i2} \to \bigvee_{j=1}^m t_{j1} = t_{j2})$. Assume that $\mathcal{V}$ satisfies condition (K). Then $\mathcal{V} \models \phi$ iff for every $X \in \mathcal{K}$, $\mathcal{H}(X) \models \phi$.*

Proof: *(Sketch)* The direct implication follows from the fact that, by (K)(ii), for every $X \in \mathcal{K}$, $\mathcal{H}(X) \in \mathcal{V}$; the inverse implication follows from the fact that, by (K)(i), for every $A \in \mathcal{V}$, the *RT* Σ-relational structure corresponding to $D(A)$ is in $\mathcal{K}$, and that, by the Priestley representation theorem, A is isomorphic to $\mathsf{ClopenOF}(D(A))$ which is a subalgebra of $\mathcal{O}(D(A))$. □

3.3 Structure-Preserving Translation to Clause Form

If the class $\mathcal{K}$ is first-order definable, Theorem 2 justifies a structure-preserving translation of universal formulae to sets of clauses, inspired by the method of Tseitin [22] for transforming quantifier-free formulae to clausal normal form.

Theorem 3. *Assume that $\mathcal{V}$ satisfies (K), where the class $\mathcal{K}$ is definable by a finite set C of first-order sentences*[1]. *Let $\phi = \forall x_1, \dots, x_k(\bigwedge_{i=1}^n s_{i1} = s_{i2} \rightarrow \bigvee_{j=1}^m t_{j1} = t_{j2})$. Then $\mathcal{V} \models \phi$ iff the following conjunction is unsatisfiable:*

$$\left\{\begin{array}{ll}
(\mathsf{Dom}) & C, \\
(\mathsf{Her}) & \forall x, y(x \leq y \wedge P_e(x) \rightarrow P_e(y)), \\
(\mathsf{Ren}) & \\
\quad (1,0) & \forall x P_1(x), \ \textit{resp.}\ \forall x \neg P_0(x), \\
\quad (\wedge) & \forall x(P_{e_1 \wedge e_2}(x) \leftrightarrow P_{e_1}(x) \wedge P_{e_2}(x)), \\
\quad (\vee) & \forall x(P_{e_1 \vee e_2}(x) \leftrightarrow P_{e_1}(x) \vee P_{e_2}(x)), \\
\quad (Lh) & \forall x(P_{h(e)}(x) \leftrightarrow P_e(\overline{h}(x))), \\
\quad (La) & \forall x(P_{k(e)}(x) \leftrightarrow \neg P_e(\overline{k}(x))), \\
\quad (Jh) & \forall x(P_{f(e_1,\dots,e_p)}(x) \leftrightarrow \exists x_1, \dots, x_p(\bigwedge_{i=1}^p P_{e_i}(x_i) \wedge \overline{f}(x_1, \dots, x_p, x))), \\
\quad (Mh) & \forall x(P_{g(e_1,\dots,e_p)}(x) \leftrightarrow \forall x_1, \dots, x_p(\overline{g}(x_1, \dots, x_p, x) \rightarrow (\bigvee_{i=1}^p P_{e_i}(x_i)))), \\
\quad (Ja) & \forall x(P_{f(e_1,\dots,e_p)}(x) \leftrightarrow \exists x_1, \dots, x_p(\bigwedge_{i=1}^p \neg P_{e_i}(x_i) \wedge \overline{f}(x_1, \dots, x_p, x))), \\
\quad (Ma) & \forall x(P_{g(e_1,\dots,e_p)}(x) \leftrightarrow \forall x_1, \dots, x_p(\overline{g}(x_1, \dots, x_p, x) \rightarrow (\bigvee_{i=1}^p \neg P_{e_i}(x_i)))), \\
\quad (\Rightarrow) & \forall x(P_{e_1 \Rightarrow e_2}(x) \leftrightarrow \forall y(x \leq y \wedge P_{e_1}(y) \rightarrow P_{e_2}(y))), \\
\quad (\neg) & \forall x(P_{\neg e}(x) \leftrightarrow \forall y(x \leq y \rightarrow \neg P_e(y))), \\
(\mathsf{P}) & \forall x(\bigwedge_{i=1}^n P_{s_{i1}}(x) \leftrightarrow P_{s_{i2}}(x)), \\
(\mathsf{N_1}) & \exists x_1 P_{t_{11}}(x_1) \not\leftrightarrow P_{t_{12}}(x_1), \\
\dots & \dots \\
(\mathsf{N_m}) & \exists x_m P_{t_{m1}}(x_m) \not\leftrightarrow P_{t_{m2}}(x_m),
\end{array}\right.$$

where the unary predicates P_e are indexed by elements in $ST(\phi)$.

Proof: *(Sketch)* By Theorem 2, $\mathcal{V} \models \phi$ iff for every $X \in \mathcal{K}$ and every $m : \{x_1, \dots, x_k\} \rightarrow \mathcal{H}(X)$, $\mathcal{H}(X) \models_m \phi$. The conclusion now follows from the fact the set of formulae $(\mathsf{Dom}) \cup (\mathsf{Her}) \cup (\mathsf{Ren}) \cup (\mathsf{P}) \cup (\mathsf{N_1}) \cup \dots \cup (\mathsf{N_m})$ is satisfiable iff there exists $X \in \mathcal{K}$ and $m : \{x_1, \dots, x_k\} \rightarrow \mathcal{H}(X)$ such that $\mathcal{H}(X) \not\models_m \phi$. □

The problem of deciding whether a universal formula is true in a variety $\mathcal{V}$ can be reduced to deciding whether a set of clauses corresponding to the conjunction in Theorem 3 is unsatisfiable. In what follows we show that ordered chaining with selection gives a decision procedure in the case when $\mathcal{V}$ is the variety DLO_Σ, the variety of distributive p-lattices or that of Heyting algebras.

[1] The set C contains formulae expressing the properties of $\leq$ (such as reflexivity and transitivity), monotonicity properties of the functions and relations in Σ, as well as the possible interdependence between the functions and relations in $\Sigma \cup \{\leq\}$

3.4 DLO_Σ: Decidability and Complexity Results

Let now $\mathcal{V} = \mathsf{DLO}_\Sigma$. From the results on Priestley duality for DLO_Σ and by Theorem 1 it follows that DLO_Σ satisfies condition (K) where $\mathcal{K} = RTS_\Sigma$. This class is defined by a set RT of formulae expressing the reflexivity and transitivity of $\leq$, together with the set C_Σ of formulae, corresponding to the fact that in every structure in RTS_Σ the functions in Lh preserve $\leq$, those in La reverse $\leq$, the relations in $Jh \cup Ja$ are increasing and those in $Mh \cup Ma$ are decreasing:

C_{Lh}	$\forall x, y(x \leq y \rightarrow \overline{h}(x) \leq \overline{h}(y))$	$h \in Lh$,
C_{La}	$\forall x, y(x \leq y \rightarrow \overline{k}(y) \leq \overline{k}(x))$	$k \in La$,
$C_{Jh,Ja}$	$\forall x_1, \dots, x_p, x, y(x \leq y \wedge \overline{f}(x_1, \dots, x_p, x) \rightarrow \overline{f}(x_1, \dots, x_p, y))$	$f \in Jh \cup Ja$,
$C_{Mh,Ma}$	$\forall x_1, \dots, x_p, x, y(y \leq x \wedge \overline{g}(x_1, \dots, x_p, x) \rightarrow \overline{g}(x_1, \dots, x_p, y))$	$g \in Mh \cup Ma$.

The set $\mathcal{C}_\Sigma(\phi)$ of clauses generated by translating the conjunction in Theorem 3 to clause form is indicated below. (Note that $|\mathcal{C}_\Sigma(\phi)| = \mathcal{O}(\mathsf{length}(\phi))$.)

(Dom) clause form of the formulae in C_Σ,
(RT) clause form of the reflexivity and transitivity axioms,
(Her) $\{\neg x \leq y, \neg P_e(x), P_e(y)\}$,
(Ren)
  (1,0) $\{P_1(x)\}, \{\neg P_0(x)\}$,
  ($\wedge$) $\{\neg P_{e_1 \wedge e_2}(x), P_{e_1}(x)\}, \{\neg P_{e_1 \wedge e_2}(x), P_{e_2}(x)\}, \{\neg P_{e_1}(x), \neg P_{e_2}(x), P_{e_1 \wedge e_2}(x)\}$,
  ($\vee$) $\{\neg P_{e_1 \vee e_2}(x), P_{e_1}(x), P_{e_2}(x)\}, \{\neg P_{e_1}(x), P_{e_1 \vee e_2}(x)\}, \{\neg P_{e_2}(x), P_{e_1 \vee e_2}(x)\}$,
  (Lh) $\{\neg P_{h(e)}(x), P_e(\overline{h}(x))\}, \{P_{h(e)}(x), \neg P_e(\overline{h}(x))\}$,
  (La) $\{P_{k(e)}(x), P_e(\overline{k}(x))\}, \{\neg P_{k(e)}(x), \neg P_e(\overline{k}(x))\}$,
  (Jh_1) $\{\neg P_{f(e_1,\dots,e_p)}(x), P_{e_i}(c_i^{f(e_1,\dots,e_p)}(x))\}, i = 1, \dots, p$,
  (Jh_2) $\{\neg P_{f(e_1,\dots,e_p)}(x), \overline{f}(c_1^{f(e_1,\dots,e_p)}(x), \dots, c_p^{f(e_1,\dots,e_p)}(x), x)\}$,
  (Jh_3) $\{P_{f(e_1,\dots,e_p)}(x), \neg P_{e_1}(y_1), \dots, \neg P_{e_p}(y_p), \neg \overline{f}(y_1, \dots, y_p, x)\}$,
  (Mh_1) $\{P_{g(e_1,\dots,e_p)}(x), \neg P_{e_i}(c_i^{g(e_1,\dots,e_p)}(x))\}, i = 1, \dots, p$,
  (Mh_2) $\{P_{g(e_1,\dots,e_p)}(x), \overline{g}(c_1^{g(e_1,\dots,e_p)}(x), \dots, c_p^{g(e_1,\dots,e_p)}(x), x)\}$,
  (Mh_3) $\{\neg P_{g(e_1,\dots,e_p)}(x), P_{e_1}(y_1), \dots, P_{e_p}(y_p), \neg \overline{g}(y_1, \dots, y_p, x)\}$,
  (Ja_1) $\{\neg P_{f(e_1,\dots,e_p)}(x), \neg P_{e_i}(c_i^{f(e_1,\dots,e_p)}(x))\}, i = 1, \dots, p$,
  (Ja_2) $\{\neg P_{f(e_1,\dots,e_p)}(x), \overline{f}(c_1^{f(e_1,\dots,e_p)}(x), \dots, c_p^{f(e_1,\dots,e_p)}(x), x)\}$,
  (Ja_3) $\{P_{f(e_1,\dots,e_p)}(x), P_{e_1}(y_1), \dots, P_{e_p}(y_p), \neg \overline{f}(y_1, \dots, y_p, x)\}$,
  (Ma_1) $\{P_{g(e_1,\dots,e_p)}(x), P_{e_i}(c_i^{g(e_1,\dots,e_p)}(x))\}, i = 1, \dots, p$,
  (Ma_2) $\{P_{g(e_1,\dots,e_p)}(x), \overline{g}(c_1^{g(e_1,\dots,e_p)}(x), \dots, c_p^{g(e_1,\dots,e_p)}(x), x)\}$,
  (Ma_3) $\{\neg P_{g(e_1,\dots,e_p)}(x), \neg P_{e_1}(y_1), \dots, \neg P_{e_p}(y_p), \neg \overline{g}(y_1, \dots, y_p, x)\}$,
(P) $\{\neg P_{s_{i1}}(x), P_{s_{i2}}(x)\}, \{P_{s_{i1}}(x), \neg P_{s_{i2}}(x)\}, i = 1, \dots, n$,
(N) $\{P_{t_{j1}}(c_j), P_{t_{j2}}(c_j)\}, \{\neg P_{t_{j1}}(c_j), \neg P_{t_{j2}}(c_j)\}, j = 1, \dots, m$,

where the predicate symbols P_e are indexed by subterms in $ST(\phi)$, $c_i^{f(e_1,\dots,e_p)}$ are Skolem functions obtained from the existential quantifiers in the transformation of terms of the form $f(e_1, \dots, e_p)$, where $p = a(f)$; $c_1, \dots, c_m$ are Skolem constants introduced by the existential quantifiers in $(\mathsf{N}_1), \dots, (\mathsf{N}_\mathsf{m})$ in Theorem 3; and $\overline{f}, \overline{g}$ for $f \in Jh \cup Ja, g \in Mh \cup Ma$ are also considered predicate symbols.

The following result is a direct consequence of Theorem 3.

Corollary 1. $\mathsf{DLO}_\Sigma \models \phi$ *iff* $\mathcal{C}_\Sigma(\phi)$ *is unsatisfiable.*

We now show that ordered chaining with selection is a decision procedure for $\mathcal{C}_\Sigma(\phi)$. We assume given a reduction ordering $\succ$ which is total on ground terms. Based on $\succ$, an ordering on literals (also denoted by $\succ$) will be defined. Let c be the complexity measure defined for every ground literal L by $c_L = (\mathsf{max}_L, p_L, s_L)$ where max_L is the maximal term occurring in L, p_L is 1 if L is negative and 0 if L is positive, and s_L is 1 if L is of the form $(\neg)s \leq t$ with $s \succ t$, and 0 otherwise. (The choice of c_L was inspired by [8].) c induces a well-founded ordering $\succ_c$ on ground literals, defined by $L \succ_c L'$ iff $c_L > c_{L'}$ (in the lexicographic combination of $\succ$ and $>$, where $1 > 0$). Let $\succ$ be a total and well-founded extension of $\succ_c$. (Such an ordering is left-to-right admissible in the sense used in [2].) Let S be the selection function that selects (i) all negative occurrences of literals containing $\leq$, and (ii) all occurrences of negative literals containing a predicate symbol in $Jh \cup \dots \cup Ma$ in clauses which do not contain $\leq$. The chaining calculus based on the literal ordering $\succ$ and the selection function S will be denoted $\mathsf{C}_S^{\succ}$.

Theorem 4. *$\mathsf{C}_S^{\succ}$ decides the unsatisfiability of $\mathcal{C}_\Sigma(\phi)$ in exponential time.*

Proof: *(Sketch)* It can be shown that, due to ordering constraints and the choice of S, no $\mathsf{C}_S^{\succ}$ inferences between clauses in $(\mathsf{RT}) \cup (\mathsf{Her})$ and clauses in $(\mathsf{Ren}) \cup (\mathsf{P}) \cup (\mathsf{N})$ are possible, and all clauses obtained by $\mathsf{C}_S^{\succ}$ inferences from $(\mathsf{RT}) \cup (\mathsf{Her})$ are redundant. Using the definition of $\succ$ on literals, it can be shown that all clauses obtained by ordered resolution with selection from $(\mathsf{Ren}) \cup (\mathsf{P}) \cup (\mathsf{N})$ have term depth 1 and either (i) are ground (and contain only one constant), or (ii) contain only one variable (occurring in every literal) and no constant or, (iii) are factors of (Jh_3), (Ja_3), (Mh_3) or (Ma_3). Moreover, all negative occurrences of a predicate symbol in $Jh \cup Ja \cup Mh \cup Ma$ must occur in clauses of type (iii). Due to the definition of $\succ$, neither the term depth of clauses nor the number of variables in the clause increase by ordered resolution. For every constant c_i (resp. every variable x) the number of all possible atoms for the clauses containing c_i (resp. x) and of term depth at most 1 is $ns \cdot (mp \cdot ns + nf + 1) + np \cdot (mp \cdot ns + nf + 1)^{mp+1}$ (ns resp. np is the number of all unary, resp. at most mp-ary predicate symbols; among the function symbols one also has to count the (unary) Skolem functions associated to the subterms in $ST(\phi)$, of which there are at most $mp \cdot ns$). This shows that, assuming np, nf, and mp are constant, the number clauses that can be generated by ordered resolution with selection from $(\mathsf{Ren}) \cup (\mathsf{P}) \cup (\mathsf{N})$ is of the order $3^{\mathcal{O}(ns^{m+1})}$. □

Remark. The above proof shows that the clauses containing the predicate symbol $\leq$ are not needed in order to prove unsatisfiability of $\mathcal{C}_\Sigma(\phi)$. The reason is that every algebra in DLO_Σ is a sublattice of a lattice whose Priestley dual has the discrete ordering, i.e. $\mathsf{DLO}_\Sigma = IS(\{L \in \mathsf{DLO}_\Sigma \mid D(L) \text{ discretely ordered}\})$, and, hence, a universal formulae is valid in DLO_Σ iff it is valid in every algebra in DLO_Σ whose dual is discretely ordered. All varieties in this subsection have this property; in Section 3.5 we discuss two varieties which do not have this property, i.e. for which $\leq$ has to be explicitly taken into account.

Example 1: The Variety D_{01} of Bounded Distributive Lattices. Let $\Sigma = \emptyset$. In this case $\mathsf{DLO}_\Sigma = \mathsf{D}_{01}$. The considerations above show that D_{01}

fulfills condition (K), $\mathcal{K}$ being the class RTS of all sets endowed with a reflexive and transitive relation. In the translation to clause form only the set $\mathcal{C}(\phi) = (\mathsf{RT}) \cup (\mathsf{Her}) \cup (\mathsf{Ren})(0,1) \cup (\mathsf{Ren})(\wedge) \cup (\mathsf{Ren})(\vee) \cup (\mathsf{P}) \cup (\mathsf{N})$ of clauses needs to be taken into account. (In this case $(\mathsf{Ren}) = (\mathsf{Ren})(0,1) \cup (\mathsf{Ren})(\wedge) \cup (\mathsf{Ren})(\vee)$.)

The results in Theorem 4 can be sharpened in this case. Due to the special form of the clauses in $\mathcal{C}(\phi)$, all possible resolvents are either ground and all literals contain the same constant, or all their literals contain the same variable (and no constant), and, additionally, the term depth of all clauses is 0. Thus, only at most $(m+1) \cdot 3^{ns}$ clauses can be generated in this case.

From the special form of the clauses in $(\mathsf{Ren}) \cup (\mathsf{P}) \cup (\mathsf{N})$ it follows that if $\mathcal{C}(\phi)$ is satisfiable, then it is satisfied by a model with m points, namely $\{c_1, \dots, c_m\}$. Moreover, $\mathcal{C}(\phi)$ is satisfiable iff there exists a $j \leq m$ such that $\mathcal{C}(\phi_j)$ (obtained from $\mathcal{C}(\phi)$ by only keeping the clauses containing c_j in (N)) is satisfied by the one point model $\{c_j\}$. This is explained by the fact that $\mathsf{D}_{01} = ISP(2)$ (the quasi-variety generated by the 2-element lattice), hence, every conditional equation is true in D_{01} iff it is true in the 2-element lattice whose Priestley dual has one element. Since D_{01} is closed under direct products, it follows [13] that $\mathsf{D}_{01} \models \phi$ iff there exists a j such that $\mathsf{D}_{01} \models \forall x_1, \dots \forall x_k (\bigwedge_{i=1}^n s_{i1} = s_{i2} \rightarrow t_{j1} = t_{j2})$ iff there exists a j such that $2 \models \forall x_1, \dots \forall x_k (\bigwedge_{i=1}^n s_{i1} = s_{i2} \rightarrow t_{j1} = t_{j2})$ iff $2^m \models \phi$. Thus, a universal formula ϕ is true in D_{01} iff it is true in 2^m, a distributive lattice whose Priestley dual has m elements and is discretely ordered.

Example 2: Bounded Distributive Lattices with Lattice (Anti)morphisms. The arguments in Theorem 4 can be adapted to bounded distributive lattices endowed with (anti)morphisms. All clauses in $(\mathsf{Ren})(0,1) \cup (\mathsf{Ren})(\wedge) \cup (\mathsf{Ren})(\vee) \cup (\mathsf{Ren})(Lh) \cup (\mathsf{Ren})(La)$ and all possible resolvents have depth at most 1 and are either ground (and all literals contain the same constant) or have exactly one variable (occurring in all literals). The number of all function symbols is in this case nf (no Skolem functions occur). Therefore, at most $(m+1) \cdot 3^{ns \cdot (nf+1)}$ different clauses can be generated.

The fact that a universal formulae is valid in DLO_Σ iff it is valid in every algebra in DLO_Σ whose dual is discretely ordered, opens the way for further results.

Proposition 1. *The satisfiability problem for* $\phi = \forall x_1, \dots, x_k (\bigwedge_{i=1}^n s_{i1} = s_{i2} \rightarrow \bigvee_{j=1}^m t_{j1} = t_{j2})$ *can be reduced to the satisfiability problem for the monadic class with equality in polynomial time w.r.t. the length of* ϕ.

Proof: *(Sketch)* The clauses in $(\mathsf{Ren}) \cup (\mathsf{P}) \cup (\mathsf{N})$ can be brought to the form of *flat clauses* considered in [3]. This can be done in the following steps:

1. Replace every occurrence of a literal of the form $\overline{f}(t_1, \dots, t_p)$ or $\neg\overline{f}(t_1, \dots, t_p)$ with $\overline{f}(t_1, \dots, t_p) = \top$, resp. $\overline{f}(t_1, \dots, t_p) = \perp$, $f \in Jh \cup \dots \cup Ma$.
   Thus, the relation symbols in $Jh \cup Ja \cup Mh \cup Ma$ are interpreted as function symbols of a different sort (sorts can be represented by unary predicates).
2. Use variable abstraction for the clauses in $J = (Jh_2) \cup (Ja_2)$ and $M = (Mh_2) \cup (Ma_2)$, to bring them in the following form:

(J') $\{\neg P_{f(e_1,\dots,e_p)}(x), y_1 \neq c_1^{f(e_1,\dots,e_p)}(x), \dots, y_p \neq c_p^{f(e_1,\dots,e_p)}(x), \overline{f}(y_1, \dots, y_p, x) = \top\}$

(M') $\{P_{g(e_1,\dots,e_p)}(x), y_1 \neq c_1^{g(e_1,\dots,e_p)}(x), \dots, y_p \neq c_p^{g(e_1,\dots,e_p)}(x), \overline{g}(y_1, \dots, y_p, x) = \top\}$

The set of clauses obtained this way can be regarded as the result of skolemizing a formula $\overline{\phi}$ (in prenex form) in the monadic class with equality. The translation to clause form, the procedure above, and $\mathsf{length}(\overline{\phi})$ are polynomial w.r.t. $\mathsf{length}(\phi)$. □

Superposition with simplification is a decision procedure for the monadic class with equality [3]. The reduction to the monadic class with equality also offers decidability and complexity results for those subvarieties of DLO_Σ in which (i) the conditions in (Dom) are either (a) in C_Σ or (b) expressible in the monadic class with equality, and (ii) in case (b), only $=$ and the predicate symbols corresponding to relations in $Jh \cup \dots \cup Ma$ may occur. An upper bound for the decision problem for the monadic class with equality is NEXPTIME (cf. e.g. [3]). This gives an upper bound for the complexity of the universal Horn theory of such varieties.

3.5 Distributive *p*-Lattices and Heyting Algebras

Let B_ω be the variety of distributive p-lattices, and let H be the variety of Heyting algebras. From the Priestley duality for distributive p-lattices and Heyting algebras and from Theorem 1 it follows that both B_ω and H fulfill condition (K), with $\mathcal{K} = RTS$, i.e. (i) for every $A \in \mathsf{B}_\omega$ or H, $D(A) \in RTS$ (if the topology is ignored); and (ii) for every $(X, \leq) \in RTS$, $(\mathcal{H}(X), \cup, \cap, \neg, \emptyset, X) \in \mathsf{B}_\omega$ and $(\mathcal{H}(X), \cup, \cap, \Rightarrow, \neg, \emptyset, X) \in \mathsf{H}$, where $\neg$ and $\Rightarrow$ are as defined in Theorem 1(5).

Let $\phi = \forall x_1, \dots, x_k(\bigwedge_{i=1}^n s_{i1} = s_{i2} \rightarrow \bigvee_{j=1}^m t_{j1} = t_{j2})$. We reduce the problem of deciding whether $\mathcal{V} \models \phi$ to a problem solved in [8]. By the result of McKinsey mentioned before, $\mathcal{V} \models \phi$ iff $\mathcal{V} \models \phi_j$ for some j, where $\phi_j = \forall x_1 \dots x_k(\bigwedge_{i=1}^n s_{i1} = s_{i2} \rightarrow t_{j1} = t_{j2})$. So the problem of deciding $\mathcal{V} \models \phi$ reduces to deciding $\mathcal{V} \models \phi_j$ for $j = 1, \dots, m$. By Theorem 3, $\mathcal{V} \models \phi_j$ iff the set of clauses $\mathcal{C}(\phi_j)$ is unsatisfiable, where $\mathcal{C}(\phi_j)$ is obtained by adjoining to $(\mathsf{RT}) \cup (\mathsf{Her}) \cup (\mathsf{Ren})(\wedge) \cup (\mathsf{Ren})(\vee) \cup (\mathsf{P}) \cup (\mathsf{N})_\mathsf{j}$ the clauses corresponding to $(\mathsf{Ren})(\neg)$ if $\mathcal{V} = \mathsf{B}_\omega$, respectively to $(\mathsf{Ren})(\neg)$ and $(\mathsf{Ren})(\Rightarrow)$ if $\mathcal{V} = \mathsf{H}$ (where $(\mathsf{Ren})(\neg)$ and $(\mathsf{Ren})(\Rightarrow)$ are as in Theorem 3, and $(\mathsf{N})_\mathsf{j}$ is (N) for c_j only).

Let $\mathsf{C}_S^\succ$ be the chaining calculus where $\succ$ is a total, well-founded ordering on ground literals compatible with the complexity measure c_L defined in Section 3.4 (hence left-to-right admissible [2]), and, if a clause C contains a literal of the form $\neg s \leq t$ with $s \succeq t$, the selection function S selects one such literal.

Theorem 5. *For every $j = 1, \dots, m$, $\mathsf{C}_S^\succ$ (with eager condensation) decides the unsatisfiability of $\mathcal{C}(\phi_j)$.*

Proof: *(Sketch)* The set $\mathcal{C}(\phi_j)$ is in the class of clauses considered in [8]. There it is proved that $\mathsf{C}_S^\succ$ with eager condensation is a decision procedure for this kind of clauses. (We use the fact that $\mathcal{C}(\phi_j)$ has one constant; if $m > 1$, the existence of m constants may cause problems in adapting Lemma 2 in [8].) The complexity of the method is doubly exponential; a single-exponential space complexity can be obtained by splitting the clauses into their variable-disjoint regions and connecting them with the help of auxiliary monadic predicates as pointed out in [8]. □

4 Experiments

We present some concrete, relatively simple examples which illustrate the type of problems that can be solved with the method described in this paper (**RTS**), and the way this method compares to a more standard approach, (**DLat**), that proves that the conjunction of the negation of the formulae above and the axioms for bounded distributive lattices with operators is unsatisfiable (in first-order logic with equality). We considered the following formulae:

- $\phi_1 = \forall a\, \forall b\, \forall c\, (a \leq b \rightarrow a \vee (c \wedge b) = (a \vee c) \wedge b)$,
- $\phi_2 = \forall a\, \forall b\, \forall c\, ((a \wedge b = c \wedge b \,\&\, a \vee b = c \vee b) \rightarrow a = c)$,
- $\phi_3 = \forall a\, \forall b\, \forall c\, ((k^2(a) \leq a \vee k(a) \,\&\, k^3(b) = a \vee k(a) \,\&\, k^2(a) \leq k(a) \vee k(b) \vee k(c) \,\&\, k^3(b) \leq k(a) \vee k(b) \vee k(c)) \rightarrow k^2(a \vee k(b)) \leq (a \wedge k(b \wedge c)) \vee k(a))$, $k \in La$,
- $\phi_4 = \forall a\, \forall b\, f(k(a \vee b)) = f(k(a)) \vee f(k(b))$, where $f \in Ja$ and $k \in La$,
- $\phi_5 = \forall a\, \forall b\, \forall c\, \forall d\, ((f(a \vee b, d) = f(c \vee b, d) \,\&\, f(a, d) \wedge f(b, d) = f(c, d) \wedge f(b, d)) \rightarrow f(a, d) = f(c, d))$, where $f \in Jh$.

The translation to clause form in **RTS** used the results in Theorem 2 and Theorem 3. According to the proof of Theorem 4, all clauses containing $\leq$ were ignored. In addition, to reduce the number of clauses generated, an inequality $a \leq b$ was directly replaced by $\forall x (P_a(x) \rightarrow P_b(x))$. In **DLat** we experimented with various axioms for distributivity, namely (j) joins over meets, (m) meets over joins, and (b) both. The unsatisfiability of the resulting sets of clauses was checked by SPASS [23]. In both cases we indicate the number of input and derived clauses, memory and time needed by SPASS V0.92 (on a 200 MHz Pentium Pro).

Formula	**Variety**	**RTS**				**DLat**				
		# Cl (in)	# Cl (der)	Mem (KB)	Time (ms)		# Cl (in)	# Cl (der)	Mem (KB)	Time (ms)
						j	13	1	382	20
ϕ_1	D_{01}	15	19	436	30	m	13	354	590	180
						b	14	465	633	230
						j	13	2	383	30
ϕ_2	D_{01}	18	31	421	20	m	14	3347	2388	4770
						b	15	5533	3609	11990
						j	20	4042	3532	10860
ϕ_3	DLO_{La}	43	28	448	30	m	20			∞
						b	21			∞
						j	18	1	386	20
ϕ_4	$\mathsf{DLO}_{La,Ja}$	23	44	450	80	m	18	1	386	20
						b	19	0	385	10
						j	18	5703	4922	16150
ϕ_5	DLO_{Jh}	38	72	470	90	m	18	5341	4541	15100
						b	19			∞

#Cl (in) resp. (der) represents the number of input, resp. derived clauses, and ∞ indicates the fact that execution did not terminate after more than 3 min.

The results above suggest that, except for very regular and simple formulae, or for purely equational formulae, the first method, based on results presented in this paper, behaves better than the second. In the future we plan to analyze more complex examples. We would be also interested to compare the theoretical complexity of our method with that of other methods.

5 Conclusions and Plans for Future Work

In this paper we presented a resolution-based method for automated theorem proving in the universal theory of certain varieties of distributive lattices with operators. The method is based on extensions of the Priestley representation theorem to distributive lattices with operators. Based on it, we obtained decidability and complexity results (upper bounds) for the universal word problem of $\mathsf{D}_{01}, \mathsf{DLO}_{\Sigma}$, and for the variety of distributive p-algebras and that of Heyting algebras. The complexity results agree with those established for (boolean) Tarskian set constraints without functions in [12], but the methods we use are different. The fact that the same type of structures are used as relational models for distributive lattices, distributive p-lattices and Heyting algebras (the only difference is the signature) shows that the restriction of the universal theory of Heyting algebras (or distributive p-lattices) to the signature $\{0, 1, \vee, \wedge\}$ coincides with the universal theory of distributive lattices. This remark is consistent with the remarks in [21] on the similarity of the cut rules necessary for the calculus for distributive lattices developed there and the cut rules in intuitionistic logic.

By analyzing the possible inferences in a suitably chosen ordered chaining calculus, we obtained a better understanding of the structure of such varieties.

These results seem to open a promising field of research that we would like to explore in future work. We expect to be able to use similar ideas for other varieties of distributive lattices or Heyting algebras with operators. One problem to be solved is to find conditions for such varieties that would give decidability results. It would also be important to find conditions which, given a variety $\mathcal{V}$ of distributive lattices with operators, ensure that a class $\mathcal{K}$ of (first-order definable) relational structures can be found, such that condition (K) is satisfied.

Acknowledgments. I thank Harald Ganzinger, Christoph Meyer and Uwe Waldmann for helpful discussions. In particular, I thank Harald Ganzinger for drawing my attention to the link with existing results on Tarskian set constraints. I also thank the referees for their helpful comments.

References

1. L. Bachmair and H. Ganzinger. Completion of first-order clauses with equality by strict superposition. In St. Kaplan and M. Okada, editors, *Proc. 2nd International Workshop on Conditional and Typed Rewriting,* Montreal, LNCS 516, pages 162–180, Berlin, 1991. Springer-Verlag.
2. L. Bachmair and H. Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *Journal of the ACM*, 45(6):1007–1049, 1998.
3. L. Bachmair, H. Ganzinger, and U. Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Computational Logic and Proof Theory, 3rd Kurt Gödel Colloquium*, LNCS 713, pages 83–96. Springer Verlag, 1993.

4. S. Burris and R. McKenzie. *Decidability and Boolean Representations.* Memoirs of the AMS, Volume 32, Number 246. American Mathematical Society, 1981.
5. S. Burris and H.P. Sankappanavar. *A Course in Universal Algebra.* Graduate Texts in Mathematics. Springer, 1981.
6. S.S. Cosmadakis. *Equational Theories and Database Constraints.* PhD thesis, Massachusetts Institute of Technology, 1985.
7. B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.
8. H. Ganzinger, U. Hustadt, C. Meyer, and R. Schmidt. A resolution-based decision procedure for extensions of K4. Proceedings of AIML'98, 1999. To appear.
9. R. Goldblatt. Varieties of complex algebras. *Annals of Pure and Applied Logic*, 44(3):153–301, 1989.
10. H.B. Hunt, D.J. Rosenkrantz, and P.A. Bloniarz. On the computational complexity of algebra of lattices. *SIAM Journal of Computation*, 16(1):129–148, 1987.
11. B. Jónsson and A. Tarski. Boolean algebras with operators, Part I&II. *American Journal of Mathematics*, 73&74:891–939&127–162, 1951&1952.
12. D. McAllester, R. Givan, D. Kozen, and C. Witty. Tarskian set constraints. In *Proceedings of the Eleventh Annual IEEE Symposium on Logic In Computer Science*, pages 138–151. IEEE Computer Society Press, 1996.
13. J.C.C. McKinsey. The decision problem for some classes of sentences without quantifiers. *The Journal of Symbolic Logic*, 8(3):61–76, 1943.
14. P. Mielniczuk and L. Pacholski. Tarskian set constraints are in NEXPTIME. In Lubos Prim et. al., editor, *Proceedings of MFCS'98*, LNCS 1450, pages 589–596. Springer Verlag, 1998.
15. H. Priestley. Ordered sets and duality for distributive lattices. *Annals of Discrete Mathematics*, 23:39–60, 1984.
16. H.A. Priestley. Representation of distributive lattices by means of ordered Stone spaces. *Bull. London Math. Soc.*, 2:186–190, 1970.
17. T. Skolem. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit und Beweisbarkeit mathematischen Sätze nebst einem Theoreme über dichte Mengen. *Videnskapsselskapets skrifter I, Matematisk-naturvidenskabelig klasse, Videnskabsakademiet i Kristiania*, 4:1–36, 1920. Also in T. Skolem, Select Works in Logic, Scandinavian University Books, Oslo, 1970, pp.103–136.
18. V. Sofronie-Stokkermans. *Fibered Structures and Applications to Automated Theorem Proving in Certain Classes of Finitely-Valued Logics and to Modeling Interacting Systems.* PhD thesis, RISC-Linz, J.Kepler University Linz, Austria, 1997.
19. V. Sofronie-Stokkermans. Duality and canonical extensions of bounded distributive lattices with operators, and applications to the semantics of non-classical logics I, II. *Studia Logica*, 1999. To appear.
20. V. Sofronie-Stokkermans. Representation theorems and theorem proving in non-classical logics. In *Proceedings of the 29th IEEE International Symposium on Multiple-Valued Logic.* IEEE Computer Sociaty Press, 1999. To appear.
21. G. Struth. *Canonical Transformations in Algebra, Universal Algebra, and Logic.* PhD thesis, Max-Planck-Institut für Informatik, Saarbrücken, 1998.
22. G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Seminars in Mathematics V.A. Steklov Math. Inst., Leningrad*, volume 8, pages 115–125. Consultants Bureau, New York-London, 1970. Reprinted in J. Siekmann and G. Wrightson (eds): Automation of Reasoning, Vol. 2, 1983, Springer, pp.466–486.
23. Ch. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER Version 0.42. In M.A. McRobie and J.K. Slaney, editors, *Proceedings of CADE-13*, LNCS 1104, pages 141–145. Springer Verlag, 1996.

Maslov's Class K Revisited

Ullrich Hustadt and Renate A. Schmidt

Department of Computing, Manchester Metropolitan University
Chester Street, Manchester M1 5GD, United Kingdom
{U.Hustadt,R.A.Schmidt}@doc.mmu.ac.uk

Abstract. This paper gives a new treatment of Maslov's class K in the framework of resolution. More specifically, we show that K and the class DK consisting of disjunction of formulae in K can be decided by a resolution refinement based on liftable orderings. We also discuss relationships to other solvable and unsolvable classes.

1 Introduction

Maslov's class K [13] is one of the most important solvable fragments of first-order logic. It contains a variety of classical solvable fragments including the Monadic class, the initially extended Skolem class, the Gödel class, and the two-variable fragment of first-order logic FO^2 [4]. It also encompasses a range of non-classical logics, like a number of extended modal logics, many description logics used in the field of knowledge representation [11,4, chap. 7], and some reducts of representable relational algebras.

For this reason practical decision procedures for the class K are of general interest. According to Maslov [13] the inverse method provides a means to decide the validity of disjunctions of formulae in the class K. Although Kuehner [12] noted in 1971 that there is a one-to-one correspondence between derivations in the inverse method and resolution, only in 1993 a decision procedure for a subclass of the dual of K based on a refinement of resolution is described by Zamov [4, chap. 6]. His techniques are based on non-liftable orderings which have limitations regarding the application of some standard simplification rules and the completeness proof relies on a π-ordering. An uncertainty regarding completeness (see [15]) is clarified by the work of de Nivelle [3].

In this paper we are concerned with the problem of deciding satisfiability for the dual class of K, which we call $\overline{\mathrm{K}}$, and also the class $\overline{\mathrm{DK}}$ consisting of conjunctions of formulae in $\overline{\mathrm{K}}$. In the context of resolution-based decision procedures these classes are of particular interest, since even ordering refinements of resolution do not prevent the growth of term and variable height. Most of the results in the literature on decision procedures based on ordered resolution consider classes where (i) every non-ground term contains all the variables of a clause and either (ii) all literals share the same variables, or (iii) it is possible to identify a literal either by polarity or maximal height that contains all the variables of a clause. Even unrestricted resolution, and factoring, preserve properties (i) and (ii). Then an atom ordering can be utilised to ensure that for all clauses in a derivation the maximal height of variable occurrences does not increase and the height of literals is bounded [3,4, chap. 4]. Consequently, any derivation terminates. In the case (i) and (iii), a selection function and an atom ordering guarantees termination and, in particular, terms do not grow [6]. A decision procedure for a subclass of $\overline{\mathrm{DK}}$ which allows term height growth, but not variable height growth, by means of a selection function is described in [11]. $\overline{\mathrm{K}}$ and $\overline{\mathrm{DK}}$ are classes where the mentioned approaches will not prevent the growth of the variable height.

This paper describes a resolution decision procedure based on liftable orderings (A-orderings) for the classes $\overline{\mathrm{K}}$ and $\overline{\mathrm{DK}}$, thereby extending the result of Zamov. Our proof is situated in the resolution framework of Bachmair and Ganzinger [1,2]. An additional renaming transformation of certain problematic clauses allows for the embedding of the classes under consideration into a class for which standard liftable orderings ensure closure under resolution and factoring as well as termination. This technique is described in Section 4. The results of Section 3 are similar to those of Zamov [4, chap. 6]. For this reason we omit the proofs. It should be noted however that our definitions of similarity and k-regularity in Section 2 are different to Zamov's, in particular, our notions allow for the presence of constants.

2 The Class $\overline{\mathrm{K}}$ and Quasi-regular Clauses

The language is assumed to be that of first-order logic without equality and without function symbols. Let ϕ be a closed formula in negation normal form and ψ a subformula of ϕ. The *ϕ-prefix* of the formula ψ is the sequence of quantifiers of ϕ which bind the free variables of ψ. If a ϕ-prefix is of the form $\exists y_1 \ldots \exists y_m \forall x_1\, Q_1 z_1 \ldots Q_n z_n$, where $m{\geq}0$, $n{\geq}0$, $Q_i \in \{\exists, \forall\}$ for all i, $1 \leq i \leq n$, then $\forall x_1\, Q_1 z_1 \ldots Q_n z_n$ is the *terminal ϕ-prefix*. For a ϕ-prefix $\exists y_1 \ldots \exists y_m$ the terminal ϕ-prefix is the empty sequence of quantifiers.

By definition, a closed formula ϕ in negation normal form belongs to the *class* $\overline{\mathrm{K}}$ if there are k quantifiers $\forall x_1, \ldots, \forall x_k$, $k{\geq}0$, in ϕ not interspersed with existential quantifiers such that for every atomic subformula ψ of ϕ the terminal ϕ-prefix of ψ

1. is either of length less than or equal to 1, or
2. ends with an existential quantifier, or
3. is of the form $\forall x_1 \forall x_2 \ldots \forall x_k$.

We say the variables $x_1, \ldots, x_k$, $k{\geq}0$, are the *fixed universally quantified variables of ϕ* and ϕ is of *grade k*, indicating the number of fixed universally quantified variables. For example, the following formulae

$$\begin{aligned}
&\phi_1 = \exists a_1 \forall x_1 \forall x_2 \exists y_1 \forall z_1 \exists y_2\, p(a_1) \wedge p(a_1, y_1) \wedge (q(x_1, a_1, x_2) \vee r(x_1, y_2, z_1))\\
&\phi_2 = \exists a_1 \exists a_2 \forall x_1 \forall x_2\, p(a_1, x_1, x_2) \vee p(x_2, a_2, x_2),
\end{aligned}$$

are elements of the class $\overline{\mathrm{K}}$ of grade 2: The variables x_1 and x_2 are the fixed universally quantified variables of ϕ_1 and ϕ_2. Every atomic subformula ψ satisfies the restrictions on the quantifier prefix binding the variables in ψ. The normal forms of Mortimer [14] for formulae in FO^2 are in $\overline{\mathrm{K}}$. Important properties of binary relations which belong to $\overline{\mathrm{K}}$ are reflexivity, irreflexivity, symmetry, seriality, and density. The following formulae

$$\begin{aligned}\phi_3 &= \forall x_1\,\forall x_2\,\forall x_3\,\neg p(x_1,x_2,x_3) \wedge q(x_1,x_2)\\ \phi_4 &= \forall x_1\,\exists x_2\,\forall x_3\,\neg p(x_1,x_2,x_3) \vee p(x_1,x_2,x_3)\end{aligned}$$

do not belong to $\overline{\mathrm{K}}$. Important properties of binary relations which cannot be expressed in $\overline{\mathrm{K}}$ are transitivity, Euclideanness, and confluence.

Since our intention is a resolution-based decision procedure for the class(es) $\overline{\mathrm{K}}$ (and $\overline{\mathrm{DK}}$) we are interested in the clause sets corresponding to formulae in $\overline{\mathrm{K}}$. Without loss of generality we can restrict ourselves to formulae in prenex form whose matrix is in conjunctive normal form, that is, formulae in $\overline{\mathrm{K}}$ have the form

$$\exists y_1 \ldots \exists y_m\, \forall x_1 \ldots \forall x_k\, Q_1 z_1 \ldots Q z_l \bigwedge_{i=1,\ldots,n} \bigvee_{j=1,\ldots,m_i} L_{i,j} \tag{1}$$

where $m{\geq}0$, $k{\geq}0$, $l{\geq}0$, $n{>}0$, $m_i{>}0$, and $L_{i,j}$ are literals. We assume that outer Skolemisation is used in the process of transforming (1) to clausal form, that is, if $\forall z_1 \ldots \forall z_p$ is the subsequence of all universal quantifiers of the ϕ-prefix of subformula $\exists z\,\phi$ of ϕ, then $\phi[z/f(z_1,\ldots,z_p)]$ is the outer Skolemisation of $\exists z\,\phi$. The class of clause sets thus obtained is denoted by $\overline{\mathrm{KC}}$.

The remainder of this section is devoted to the definition of a syntactic characterisation of the clauses in $\overline{\mathrm{KC}}$.

A term t is said to *dominate* a term s, denoted by $t \succsim_Z s$, if at least one of the following conditions is satisfied:

1. $t = s$, s is a variable,
2. $t = f(t_1,\ldots,t_n)$, s is a variable and $s = t_i$ for some i, $1 \leq i \leq n$,
3. $t = f(t_1,\ldots,t_n)$, $s = g(t_1,\ldots,t_m)$, $n \geq m \geq 0$,

The relation $\succsim_Z$ is a quasi-ordering on terms and stable with respect to substitutions on compound terms. By compound terms we mean terms which are neither constants nor variables. The relation $\succsim_Z$ is extended to sets of terms and literals as follows. The set T_1 of terms *dominates* the set T_2 of terms if for every term t_2 from T_2 there exists a term t_1 from T_1 such that t_1 dominates t_2. A literal L_1 *dominates* a literal L_2, denoted by $L_1 \succsim_Z L_2$, if the set of non-constant arguments of L_1 dominates the set of non-constant arguments of L_2. Note that $\succsim_Z$ is also a quasi-ordering on literals. We let $\sim_Z = \succsim_Z \cap \succsim_Z^{-1}$ and $\succ_Z = \succsim_Z \setminus \sim_Z$. If $s \sim_Z t$ for terms s and t, then s and t are *similar.* Analogously, for literals.

For example, the literal $p(x,y)$ dominates $q(a,x,y)$, but not $q(f(a),x,y)$. The literals $p(x,y)$, $q(a,x,y)$, and $q(y,x)$ are similar. So are $p(f(a),x)$ and $q(g(a),x)$. Note that $p(f(a),x)$ is not similar to $q(g(b),x)$, nor does one dominate the other.

Based on the quasi-ordering $\succsim_Z$ we are able to characterise a subset of the set of all terms in the following way: A term is called *regular* if it dominates all its arguments. We extend this notion to sets of terms and literals as follows.

A set of terms is called *regular* if it contains no compound terms or it contains some regular compound term which dominates all terms of this set. A literal is called *regular* if the set of its arguments is regular.

The extension of regularity to clauses, which we regard as multisets of literals, is less straightforward. We need two more definitions: A literal L is *singular* if it contains no compound term and $\mathcal{V}(L)$ is a singleton,[1] otherwise it is *non-singular*. A literal containing a compound term is *deep*, otherwise it is *shallow*. A clause C is *k-regular* if the following conditions hold:

1. C contains regular literals only.
2. k is a non-negative integer not greater than the minimal arity of function symbols occurring in C. If C does not contain compound terms, then k is arbitrary.
3. C contains some literal which dominates every literal in C.
4. If L_1 and L_2 are non-singular, shallow literals in C, then $L_1 \sim_Z L_2$.
5. If L is a non-singular, shallow literal in C, then for every compound term t occurring in C
$$\arg_{set}(L) \setminus \mathsf{F}_0 \sim_Z \arg_{set}^{1..k}(t) \setminus \mathsf{F}_0$$
holds, where $\arg_{set}(L)$ is the set of arguments of L, $\arg_{set}^{1..k}(t)$ is the set of the first k arguments of t, and F_0 is the set of all constants.

A clause is *regular* if it is k-regular for some $k{\geq}0$. A clause is called *quasi-regular* if all of its indecomposable components are regular.

For example, the clause $\{p(a, y, z), q(f(a, y, z))\}$ is 3-regular. Note that the clause $\{p(x_1, x_2, x_3)\}$ can be considered to be a 2-regular clause, although the corresponding first-order formula $\forall x_1 \forall x_2 \forall x_3 \, p(x_1, x_2, x_3)$ is of grade 3.

3 Resolution and Factoring on Quasi-regular Clauses

Next we study closure of quasi-regular clauses under resolution and factoring.

Recall that the components in the variable partition of a clause are called split components, that is, split components do not share variables. A clause which is identical to its split component is indecomposable. The condensation $\mathrm{Cond}(C)$ of a clause C is a minimal subclause of C which is a factor of C.

Theorem 1. *Every indecomposable component of a clause in the clausal form of a formula ϕ of the form (1) is k-regular.*

Lemma 2 (Properties of regular expressions [4, pages 136–144]).

1. If a regular term t dominates the term s and σ is a substitution such that $t\sigma$ is regular, then $t\sigma$ dominates $s\sigma$.
2. Let $L_1 = p(s_1, \dots, s_n)$ and $L_2 = p(t_1, \dots, t_n)$ be unifiable deep literals. If s_i is a dominating term of L_1, then also t_i is a dominating term of L_2.
3. Let L_1 and L_2 be regular literals and σ a most general unifier of L_1 and L_2. Then $L_1\sigma$ is regular.

[1] $\mathcal{V}(E)$ denotes the set of variables of an expression E.

Theorem 3. *Let $\{A_1\} \cup C_1$ and $\{\neg A_2\} \cup C_2$ be indecomposable, k-regular clauses such that A_1 and A_2 are unifiable with most general unifier σ, and A_1 and $\neg A_2$ are dominating literals in $\{A_1\} \cup C_1$ and $\{\neg A_2\} \cup C_2$, respectively. Then every split component of $(C_1 \cup C_2)\sigma$ is a k-regular clause.*

It is crucial that in Theorem 3 we require both clauses to be k-regular, for the same k, where Zamov's version only requires regularity. The resolvent of the two regular clauses $\{\neg p(g(x,y,z),z), q(x,y), r(g(x,y,z))\}$ and $\{p(g(x',y',z'), f(x'))\}$ is a counterexample for Zamov's result.

Theorem 4. *Let $\{L_1, L_2\} \cup C$ be an indecomposable, k-regular clause such that L_1 and L_2 are unifiable with most general unifier σ and L_1 is a dominating literal in $\{L_1, L_2\} \cup C$. Then $(\{L_1\} \cup C)\sigma$ is a k-regular clause.*

Lemma 5. *Let N be a set of k-regular clauses. Let n_{var} be the maximal number of distinct variables in any clause in N. Then for any clause C derivable by resolution on $\succ_Z$-maximal literals or factoring, the number of distinct variables in C is less or equal to n_{var}.*

The number of distinct regular terms over a given vocabulary is considerably smaller than the number of distinct terms based on this vocabulary.

Theorem 6. *Let N be a set of k-regular clauses. Let ar_{fun}, n_{var}, and n_{fun} be the maximal arity of function symbols, the maximal number of variables in a clause in N, and the number of function symbols in N, respectively. The number of different terms in N does not exceed*

$$n_{terms} = (n_{fun} + n_{var})^{ar_{fun}+1}.$$

Furthermore, the number of condensed clauses in N does not exceed

$$n_{clauses} = 2^{(2 \times n_{pred} \times n_{terms}^{ar_{pred}})},$$

where n_{pred} is the number of predicate symbols in N and ar_{pred} the maximal arity of a predicate symbol.

4 An Atom Ordering for Quasi-regular Clauses

An atom ordering $\succ$ is an ordering on atoms which is stable under substitution, and total (and well-founded) on ground atoms. It is extended to literals by taking the multiset extension of $\succ$ and by identifying A and $\neg B$ with the multisets $\{A\}$ and $\{B, B\}$, respectively.

Since $\succ_Z$ is not stable under substitution, we cannot construct an atom ordering based on $\succ_Z$. The first problem with $\succ_Z$ occurs on the term-level. Consider the terms $f(x,y)$ and y. Obviously, $f(x,y) \succ_Z y$ holds. However, for the substitution $\sigma = \{y/g(z)\}$, we have $f(x,y)\sigma \not\succ_Z y\sigma$. Note that $f(x,y)\sigma$ is no longer regular. According to Lemma 2(1) $t \succ_Z s$ implies $t\sigma \succ_Z s\sigma$ for any substitution σ such that t and $t\sigma$ are regular. This problem is mainly caused by the dual use of $\succ_Z$: On the one hand it is used to define the structure of regular terms, literals and clauses and on the other hand it is used to determine the literals of a clause to resolve upon. The problem can be solved by using an ordering which is compatible with $\succ_Z$ on regular terms and is stable under substitution.

The second problem with $\succ_Z$ occurs on the atom-level. As a simple example consider the atoms $p(x, y, x)$ and $p(x, x, a)$. We have that $p(x, y, x) \succ_Z p(x, x, a)$. But since the atoms have a common instance $p(a, a, a)$, there exists no atom ordering $\succ$ such that $p(x, y, x) \succ p(x, x, a)$ holds.

It is important to remember that we have to restrict resolution inference steps in a clause $\{p(x, y, x), p(x, x, a)\}$ to the first literal. For otherwise, we can no longer guarantee that resolvents of k-regular clauses are still k-regular. For example, resolution with $\{p(z, x, z), \neg p(x, x, a)\}$ (on the second literal in each clause) results in $\{p(z, x, z), p(x, y, x)\}$ which is not k-regular.

As far as the selection of suitable literals to resolve upon is concerned, clauses meeting the following two conditions cause problems.

1. The clause C contains a singular literal which has constant arguments or duplicate variable arguments.
   Suppose the opposite. Then all singular literals are monadic. All predicate symbols of shallow, non-singular literals have arity greater than one. Thus, there are no common instances of singular literals and non-singular literals. It is straightforward to define an atom ordering $\succ$ such that the singular literals are not $\succ$-maximal in C.
2. The clause C contains a shallow, non-singular literal or there is a compound term t in C such that $|\mathcal{V}(\arg_{set}^{1..k}(t))| \geq 2$.[2]
   If C contains no shallow, non-singular literals and for all compound term t in C we have that $|\mathcal{V}(\arg_{set}^{1..k}(t))| \leq 1$, then all shallow literals in C contain exactly one variable.

At first glance the second condition may seem too general. In a 2-regular clause like $\{q(f(x, y, z)), p(x, a)\}$ the literals $q(f(x, y, z))$ and $p(x, a)$ have no common instances and it is straightforward to define an atom ordering $\succ$ such that $q(f(x, y, z))$ is strictly $\succ$-maximal. However, a resolution inference step with $\{\neg q(f(x, y, z)), p(x, y)\}$ results in $\{p(x, a), p(x, y)\}$ which is the prototypical example of a problematic clause.

This analysis motivates the following definitions.

A literal L is called *CDV* (containing constants or duplicate variables) if L is singular, and there is an argument which is a constant or there are duplicate (variable) arguments. Otherwise a literal is *CDV-free.* A clause C is *CDV* if it contains a CDV literal and a shallow, non-singular literal, but no deep literal. Otherwise C is *CDV-free.*

The intuition of CDV-free clauses is that $\succ$-maximal literals are immediate for suitable atom orderings $\succ$. It should be noted that there are CDV-free clauses which contain CDV literals. Any CDV clause contains at least two literals.

[2] $|N|$ denotes the cardinality of N.

An indecomposable, k-regular clause C is *strongly CDV-free* if satisfies at least one of the following conditions.

1. C contains no CDV literal, or
2. C contains no shallow, non-singular literal and for every compound term t occurring in any literal in C, $|\mathcal{V}(\arg_{set}^{1..k}(t))| = 1$.

A set of clauses N is (strongly) *CDV-free* if every clause in N is (strongly) CDV-free.

The sample clause $\{p(f(x,y)), q(x,y), r(x,a)\}$ is CDV-free, but not strongly CDV-free. It contains the literal $r(x,a)$ which is CDV and a shallow, non-singular literal $q(x,y)$. Also the 2-regular clause $\{p(f(x,y)), r(x,a)\}$ is not strongly CDV-free: Apart from the CDV literal $r(x,a)$ it contains a deep, non-singular literal $p(f(x,y))$ such that the term $f(x,y)$ contains more than one variable. There is a subtle point to note. If we consider $\{p(f(x,y), r(x,a))\}$ as a 1-regular clause, then it is strongly CDV-free. However, in a set of 1-regular clauses, no shallow, non-singular literals occur.

The clause $\{p(x,x,a), q(x,b), p(c,x,c)\}$ is strongly CDV-free, since it contains no non-singular literal. The clause $\{p(x,y,c), q(x,y), p(x,a,y)\}$ is strongly CDV-free, since it contains no singular literal. In general, CDV-freeness does not remain invariant under resolution. A simple counterexample is a resolution inference step between $C_1 = \{p(f(x,y)), r(x,y), q(x,a)\}$ and $C_2 = \{\neg p(z)\}$ with conclusion $\{r(x,y), q(x,a)\}$. This is due to the fact that the CDV literal in C_1 is shielded by the term $f(x,y)$. Since this term is no longer present in the conclusion, the CDV literal becomes unshielded. However, C_1 is not strongly CDV-free.

We will now show that ordered factoring and ordered resolution preserve strong CDV-freeness.

Lemma 7. *Let L be a singular, CDV-free literal and σ be a substitution. Then $L\sigma$ is CDV-free.*

Proof. Let $\mathcal{V}(L) = \{x\}$. If $x \notin \mathcal{D}(\sigma)$,[3] then $L\sigma = L$ is still CDV-free. Now, assume $x \in \mathcal{D}(\sigma)$. If $x\sigma$ is a variable, then L and $L\sigma$ are identical up to the renaming of variables. So, $L\sigma$ is CDV-free. If $x\sigma$ is a constant or a compound term, then $L\sigma$ is no longer singular and therefore CDV-free. □

Theorem 8. *Let $\{L_1, L_2\} \cup C$ be an indecomposable, strongly CDV-free, k-regular clause such that L_1 and L_2 are unifiable with most general unifier σ and L_1 is a dominating literal in $\{L_1, L_2\} \cup C$. Then $(\{L_1\} \cup C)\sigma$ is strongly CDV-free.*

Proof. If $(\{L_1\} \cup C)\sigma$ is a ground clause, then it is strongly CDV-free, since it contains no CDV literals.

Suppose that $(\{L_1\} \cup C)\sigma$ is non-ground. For every literal $L\sigma$ in $(\{L_1\} \cup C)\sigma$ the set $\mathcal{V}(L\sigma)$ is a subset of $\mathcal{V}(L)$ and the height of $L\sigma$ is equal to the height of L. Thus, no singular literal L in $\{L_1, L_2\} \cup C$ will become a non-singular literal

[3] $\mathcal{D}(\sigma)$ denotes the domain of σ, and $\mathcal{C}(\sigma)$ the codomain.

$L\sigma$ in $(\{L_1\} \cup C)\sigma$, nor will any deep literal L satisfying the second condition of strong CDV-freeness turn into a deep literal $L\sigma$ violating this condition.

If $\{L_1, L_2\} \cup C$ is strongly CDV-free, since it contains no non-singular literal, the factoring inference step will not produce a non-singular literal and $(\{L_1\} \cup C)\sigma$ remains strongly CDV-free.

If $\{L_1, L_2\} \cup C$ is strongly CDV-free, since it contains no CDV literal, we can argue as follows. Suppose $(\{L_1\} \cup C)\sigma$ contains a non-ground literal $L\sigma$ which is CDV. Then L is not singular, since by Lemma 7 any singular, CDV-free literal remains CDV-free after instantiation. Also, L is not deep, since deep literals remain deep after instantiation. So, L is a shallow, non-singular literal. However, all shallow, non-singular literals in a k-regular clause contain the same set of variables and for any shallow, non-singular literal L and for all compound terms t occurring in the clause, the set of non-constant arguments of L are similar to the subset of non-constant arguments of the first k arguments of t. If instantiation with σ turns L into a singular literal with variable x, then it does so with every shallow, non-singular literal in the clause. Furthermore, the subset of non-constant terms of the first k arguments of any term t will contain only one variable, namely x. Thus, $(\{L_1\} \cup C)\sigma$ is strongly CDV-free. □

Theorem 9. *Let $\{A_1\} \cup C_1$ and $\{\neg A_2\} \cup C_2$ be indecomposable, strongly CDV-free, k-regular clauses such that A_1 and A_2 are unifiable with most general unifier σ and A_1 and $\neg A_2$ are dominating literals in $\{A_1\} \cup C_1$ and $\{\neg A_2\} \cup C_2$, respectively. Then every split component of $(C_1 \cup C_2)\sigma$ is strongly CDV-free.*

Proof. The general observation that no singular or deep literal L in one of the premises will become a shallow, non-singular literal $L\sigma$ in the conclusion $(C_1 \cup C_2)\sigma$ remains true.

We distinguish the following cases:
1. Both A_1 and $\neg A_2$ are singular literals. Then neither $\{A_1\} \cup C_1$, $\{\neg A_2\} \cup C_2$, nor $(C_1 \cup C_2)\sigma$ contain a non-singular or deep literal. So, the conclusion of the resolution inference step is strongly CDV-free.
2. A_1 is a singular literal and $\neg A_2$ is a shallow, non-singular literal. C_1 and $C_1\sigma$ contain no non-singular or deep literal and C_2 contains no deep literal. The literal $\neg A_2\sigma$ is singular. Since all shallow, non-singular literals in C_2 are similar to $\neg A_2$, $C_2\sigma$ contains no non-singular or deep literal.
3. A_1 and $\neg A_2$ are shallow, non-singular literals. Neither C_1 nor C_2 contains a deep literal. If $A_1\sigma(= A_2\sigma)$ is singular, then $(C_1 \cup C_2)\sigma$ contains no non-singular literals. Suppose $A_1\sigma$ is again a shallow, non-singular literal. Let L be a CDV-free, singular literal in either C_1 or C_2. Since $\mathcal{C}(\sigma)$ contains only variables and constants, $L\sigma$ is either ground or CDV-free, by Lemma 7.
4. A_1 is a deep literal and $\neg A_2$ is singular. $\{\neg A_2\} \cup C_2$ contains only singular literals and that $\mathcal{V}(\{\neg A_2\} \cup C_2)$ is a singleton set. So, $\neg A_2$ is not necessarily CDV-free. W.l.o.g. we can assume that σ maps the only variable occurring in $\neg A_2$ to some compound term t. This means, $\neg A_2\sigma$, and likewise any literal in $C_2\sigma$, is deep. The elements of $\mathcal{C}(\sigma_{\upharpoonright\mathcal{V}(A_1)})$ are either variables or constants. So, if L is a CDV-free literal in C_1, then $L\sigma$ is still CDV-free or ground.

Suppose C_1 contains a CDV literal. Then C_1 satisfies the second condition of strong CDV-freeness. Instantiation with σ will not introduce additional variables into compound terms and t also satisfies the requirements of the second condition. So, $(C_1 \cup C_2)\sigma$ is strongly CDV-free.

Suppose C_1 contains a shallow, non-singular literal L. Note that C_1 does not contain a CDV literal. If $L\sigma$ is still CDV-free, then $C_2\sigma$ contains no CDV literal and $(C_1 \cup C_2)\sigma$ is strongly CDV-free. If $L\sigma$ is a CDV literal, then we argue as in the proof of Theorem 8 that $(C_1 \cup C_2)\sigma$ satisfies the second condition of strong CDV-freeness.
5. A_1 is a deep literal and $\neg A_2$ is a shallow, non-singular literal. Since $C_2\sigma$ contains only deep literals and, possibly, CDV-free literals, the proof is similar to the previous case.
6. Both A_1 and $\neg A_2$ are deep literals. The important point to note in this case is the following. Let s and t be compound terms and in $(\{A_1\} \cup C_1)\sigma$ or $(\{\neg A_2\} \cup C_2)\sigma$. Let L be a shallow, non-singular literal in $\{A_1\} \cup C_1$ or $\{\neg A_2\} \cup C_2$. Then

$$\arg_{set}^{1\ldots k}(s) = \arg_{set}^{1\ldots k}(t) \quad \text{and} \quad \arg_{set}(L) \setminus \mathsf{F}_0 \sim_Z \arg_{set}^{1\ldots k}(s) \setminus \mathsf{F}_0.$$

Consequently, if A_1 is a shallow, non-singular literal in C_1 and A_2 is a shallow, non-singular literal in C_2, either both $A_1\sigma$ and $A_2\sigma$ are singular literals and $\mathcal{V}(\arg_{set}^{1\ldots k}(t))$ is a singleton set for any compound term t in $(C_1 \cup C_2)\sigma$, or neither $A_1\sigma$ nor $A_2\sigma$ are shallow, non-singular literals. It follows that $(C_1 \cup C_2)\sigma$ is strongly CDV-free. □

We have shown that the property of strong CDV-freeness is preserved under inferences by factoring and resolution on dominating literals. Since clause sets in $\overline{\text{KC}}$ are not necessarily strongly CDV-free, we define a satisfiability equivalence preserving transformation which transform any clause set N in $\overline{\text{KC}}$ into a strongly CDV-free clause set N'. The transformation is given by the rule

$$N \Rightarrow_{\mathcal{M}} N' \cup \text{Def}_L^A,$$

where (i) L is an occurrence of a CDV literal in a clause $C \in N$ which is not strongly CDV-free, (ii) A is an atom of the form $p^f(x)$ where p^f is a new predicate symbol with respect to N and x is the variable occurring in L, (iii) Def_L^A is a *definitional* clause of the form $\{\neg A, L\}$, and (iv) N' is obtained from N by replacing any occurrence of L by A.

Note that A is CDV-free and that the clause $\{\neg A, L\}$ is strongly CDV-free. As each transformation step removes at least one CDV literal in one of the clauses which are not strongly CDV-free, by a sequence of transformation steps we eventually obtain a strongly CDV-free set of clauses . We denote the resulting clause set by $N{\downarrow_{\mathcal{M}}}$.

Theorem 10. *Let N be a set of clauses. Then $N{\downarrow_{\mathcal{M}}}$ can be constructed in polynomial time, and $N{\downarrow_{\mathcal{M}}}$ is satisfiable if and only if N is satisfiable.*

An atom ordering $\succ$ suitable for our purpose has to satisfy this condition:

If a literal L is $\succ$-maximal in a clause C of a given class C, then there is no literal L' in C with $L' \succ_Z L$. (2)

We have seen that no atom ordering satisfying this condition can exist if C is the class of all (indecomposable) k-regular clauses. We will now show that if C

is the class of all (indecomposable) strongly CDV-free, k-regular clauses, we are able to define such an atom ordering.

Note that it is not relevant whether $\succ$ is applied *a priori*, like $\succ_Z$, or *a posteriori*: Since $\succ$ is stable under substitution, if $L\sigma$ is $\succ$-maximal in $C\sigma$ then L is $\succ$-maximal in C.

Let $\succ_\Sigma$ be a total precedence on predicate and functions symbols (function symbols are denoted by f and g, predicate symbols by p and q) such that

– $f \succ_\Sigma g$ if the arity of f is strictly greater than the arity of g,
– $f \succ_\Sigma p$ if f is not a constant symbol,
– $p \succ_\Sigma q$ if q is monadic and the arity of p is greater than or equal to 2, and
– $p \succ_\Sigma c$ if c is a constant symbol.

Every predicate symbol and function symbol has multiset status. Let $\succ_S$ be the recursive path ordering based on the precedence $\succ_\Sigma$.

Lemma 11. *Let L_1 and L_2 be literals in an indecomposable, strongly CDV-free, k-regular clause C. Then $L_1 \succ_Z L_2$ implies $L_1 \succ_S L_2$.*

Proof. We distinguish the following cases according to the type of L_1:
1. L_1 is non-singular and shallow. Then L_2 is singular. Therefore, L_2 contains exactly one variable x and x is an argument of L_2 and L_1. If x is the only argument of L_2, then the multiset of arguments of L_1 is obviously greater than the multiset of arguments of L_2 and the predicate symbol of L_1 has precedence over the predicate symbol of L_1 by $\succ_\Sigma$. So, $L_1 \succ_S L_2$ holds. If L_2 contains more than one argument, then L_2 is not CDV-free, contradicting our assumption that C is strongly CDV-free.
2. L_1 is deep. L_1 contains a compound term t_1 dominating all the arguments of L_1. Since $L_1 \succ_Z L_2$ and $\succ_Z$ is transitive, for every argument of t_2 of L_2, $t_1 \succ_Z t_2$ holds. The term t_2 is either a variable or a compound term. In the first case, according to the definition of $\succ_Z$, t_2 is an argument of t_1. In the second case, t_1 has the form $f(u_1, \ldots, u_m)$ and t_2 has the form $g(u_1, \ldots, u_n)$ with $m{>}n$. Then, by definition, $f \succ_\Sigma g$. Therefore, it is enough to show that $f(u_1, \ldots, u_m) \succ_S u_j$ holds, for all j with $1{\le}j{\le}n$, which is straightforward. □

5 A Decision Procedure for $\overline{\text{KC}}$

We are now ready to present a decision procedure for the class $\overline{\text{KC}}$. We adopt the resolution framework of [1,2]. The calculus is parameterised by an atom ordering $\succ$ and consists of three basic rules.

Deduce: $$\frac{N}{N \cup \{\text{Cond}(C)\}}$$

if C is either a resolvent or a factor of clauses in N.

Delete: $$\frac{N \cup \{C\}}{N}$$

if C is a tautology or N contains a clause which is a variant of C.

Split: $$\frac{N \cup \{C \cup D\}}{N \cup \{C\} \mid N \cup \{D\}}$$

if C and D are variable-disjoint.

Resolvents and factors are derived by the following rules.

Ordered Resolution: $$\frac{C \cup \{A_1\} \quad D \cup \{\neg A_2\}}{(C \cup D)\sigma}$$

where (i) σ is the most general unifier of A_1 and A_2, (ii) no literal is selected in C and $A_1\sigma$ is strictly $\succ$-maximal with respect to $C\sigma$, and (iii) $\neg A_2$ is either selected, or $\neg A_2\sigma$ is maximal in $D\sigma$ and no literal is selected in D.

Ordered Factoring: $$\frac{C \cup \{A_1, A_2\}}{(C \cup \{A_1\})\sigma}$$

where (i) σ is the most general unifier of A_1 and A_2; and (ii) no literal is selected in C and $A_1\sigma$ is $\succ$-maximal with respect to $C\sigma$.

Let R be any calculus in which (i) derivations are generated by strategies applying "Delete", "Split", and "Deduce" in this order, (ii) no application of "Deduce" with identical premises and identical consequence may occur twice on the same path in derivations, and (iii) the ordering is based on $\succ_S$ while the selection function is arbitrary.

Theorem 12. *Let N be a set of clauses in $\overline{\mathrm{KC}}$. Then,*

1. *N is unsatisfiable if and only if the R-saturation of $N{\downarrow_{\mathcal{M}}}$ contains the empty clause, and*
2. *any derivation from $N{\downarrow_{\mathcal{M}}}$ in R terminates.*

Proof. Part 1 is a direct consequence of Theorem 10 and the soundness and completeness of R [1,2].

By Theorems 3, 4, 8, and 9 the class of strongly CDV-free, k-regular clauses is closed under inference in R. Theorem 6 shows that the class of k-regular clauses is finite, provided condensation is applied eagerly. This proves part 2. □

Corollary 13. *The classes* K *and* $\overline{\mathrm{K}}$ *are decidable.*

Let us consider an example. None of the clauses in N defined below is strongly CDV-free. The transformed clause set is $N{\downarrow_{\mathcal{M}}}$.

$N:$
(1) $\{p(a,a,x), r(a,x,y)\}$
(2) $\{\neg r(a,a,x), p(a,x,y)\}$
(3) $\{r(a,a,x), \neg p(a,x,y)\}$
(4) $\{\neg p(a,a,x), \neg r(a,x,y)\}$

$N{\downarrow_{\mathcal{M}}}:$
(5) $\{p_1^+(x), r(a,x,y)\}$
(6) $\{r_1^-(x), p(a,x,y)\}$
(7) $\{r_1^+(x), \neg p(a,x,y)\}$
(8) $\{p_1^-(x), \neg r(a,x,y)\}$
(9) $\{\neg p_1^+(x), p(a,a,x)\}$
(10) $\{\neg r_1^-(x), \neg r(a,a,x)\}$
(11) $\{\neg r_1^+(x), r(a,a,x)\}$
(12) $\{\neg p_1^-(x), \neg r(a,a,x)\}$

The following presents one branch of the theorem proving derivation from $N{\downarrow_{\mathcal{M}}}$.

[(5)2, R, (8)2]	(13) $\{p_1^+(x), p_1^-(x)\}$	[(17)1, R, (8)2]	(18) $\{p_1^-(a)\}$
[(13)1, R, (9)1]	(14) $\{p(a,a,x), p_1^-(x)\}$	[(18)1, R, (12)2]	(19) $\{\neg p(a,a,a)\}$
[(14)1, R, (7)2]	(15) $\{r_1^+(a), p_1^-(x)\}$	[(19)1, R, (6)2]	(20) $\{r_1^-(a)\}$
[(15)1, Spt]	(16) $\{r_1^+(a)\}$	[(20)1, R, (10)2]	(21) $\{\neg r(a,a,a)\}$
[(16)1, R, (11)1]	(17) $\{r(a,a,a)\}$	[(21)1, R, (17)1]	(22) $\bot$

It is straightforward to check that we are able to derive the empty clause in all remaining branches by similar sequences of inference steps. Hence, the initial set N is unsatisfiable.

A particularly interesting variant of our decision procedure can by obtained by the utilisation of a selection function. The selection function $S_{\mathcal{KC}}$ selects the monadic literal $\neg A$ in a clause Def_L^A introduced by the transformation $\Rightarrow_{\mathcal{M}}$. Note that positive occurrences of A in $N{\downarrow_{\mathcal{M}}}$ are not $\succ_S$-maximal in their clauses. Thus, inferences with clauses in Def_L^A are prohibited. Only when a clause C with a $\succ_S$-maximal literal A and with selected counterpart $\neg A$ in $\mathrm{Def}_{\mathcal{M}}$ is produced by ordered resolution, will an inference step with Def_L^A be performed. Effectively, such an inference step reintroduces (an instance of) the original literal occurrence L into C. Now, leaving these inference steps aside, resolution inference steps based on the non-liftable ordering $\succ_Z$ correspond one-to-one to resolution inference step based on the liftable ordering $\succ_S$. Some factoring steps possible with respect to $\succ_Z$ are prevented in $N{\downarrow_{\mathcal{M}}}$, since one of the literals has been renamed by the transformation.

The picture changes if we take the different notions of redundancy underlying these calculi into account. The inference step leading to the clause $\{p_1^+(x), p_1^-(x)\}$ corresponds to the inference step

$$[\ (1)2,\ \mathrm{R},\ \ (4)2]\ (13')\ \{p(a,a,x), \neg p(a,a,x)\}$$

on the original clause set N which results in a tautologous clause. In a decision procedure based on the non-liftable ordering $\succ_Z$ such tautologous clauses are not redundant and, in general, cannot be eliminated without loosing completeness of the procedure. This is already evident in our example, since the only alternative inference step possible on N is the derivation of the tautologous clause $\{r(a,a,x), \neg r(a,a,x)\}$ from clauses (2) and (3). Thus, the only clauses derivable from the unsatisfiable clause set N based on the $\succ_Z$-refinement of resolution are tautologous.

In contrast, we can make use of the notion of redundancy described in [2]. For example, given the clauses $\{p(a,x,y), r(b,x,y)\}$ and $\{\neg p(a,a,z), \neg r(b,a,z)\}$ which are strongly CDV-free, the tautologous conclusion $\{p(a,a,x), \neg p(a,a,x)\}$ is redundant and can be eliminated.

6 An Extension of $\overline{\mathrm{KC}}$: The Class $\overline{\mathrm{DKC}}$

In this section we consider the class $\overline{\mathrm{DK}}$ containing all possible (finite) conjunctions of formulae of class $\overline{\mathrm{K}}$. The grade of the formulae in such a conjunction can vary. The class of clause sets obtained from formulae of class $\overline{\mathrm{DK}}$ is denoted by $\overline{\mathrm{DKC}}$.

We will prove that $\overline{\mathrm{DKC}}$ is decidable with respect to satisfiability using the procedure of the previous section.

Let ϕ be a formula of class $\overline{\mathrm{K}}$ of grade k. Let N be the corresponding set of clauses. Then we call a non-constant Skolem function f occurring in some clause in N *k-originated*. Let C be a k-regular clause such that all non-constant Skolem functions occurring in C are k-originated. Then C is *strongly k-regular.*

There is a rather subtle point to note about the definition of k-regular and strongly k-regular clauses. The value of k can be chosen almost arbitrarily if the clause does not contain compound terms. For example, if we talk about 3-regular clauses in the following, this will include clauses like $\{p(x_1, x_2), p(x_2, x_1)\}$ and $\{q(x_3, c, x_2, x_1, x_4)\}$. To distinguish clauses like these from clauses actually containing k-originated function symbols, we introduce the notion of *inhabited clauses*, that is, clauses containing at least one non-constant function symbol.

Lemma 14. *Let ϕ be a formula of class $\overline{\mathrm{K}}$ and let ϕ be of grade k. Let N be the corresponding set of clauses. Then every clause in N is strongly k-regular.*

Proof. Immediate. □

Corollary 15. *Let ϕ be a formula of class $\overline{\mathrm{DK}}$. Let N be the corresponding set of clauses. Then every clause in N is strongly k-regular for some positive k.*

Lemma 16. *Let $\{A_1\} \cup C_1$ and $\{\neg A_2\} \cup C_2$ be indecomposable, strongly k-regular clauses such that A_1 and A_2 are unifiable with most general unifier σ and A_1 and A_2 are dominating literals in $\{A_1\} \cup C_1$ and $\{\neg A_2\} \cup C_2$, respectively. Then the split components of the resolvent $(C_1 \cup C_2)\sigma$ are strongly k-regular.*

Proof. By Lemma 4 the split components of $(C_1 \cup C_2)\sigma$ are k-regular. Since all the Skolem functions of $(C_1 \cup C_2)\sigma$ already occur in one of the parent clauses, the Skolem functions are k-originated. Thus the split components of $(C_1 \cup C_2)\sigma$ ‘are strongly k-regular. □

Lemma 17. *Let C_1 be an indecomposable, strongly k-regular clause. Let C_2 be a factor of C_1. Then the split components of C_2 are strongly k-regular.*

Proof. By a similar argument to the previous lemma. □

Lemma 18. *Let $C_1 = \{A_1\} \cup D_1$ be a k_1-regular clause and $C_2 = \{\neg A_2\} \cup D_2$ a k_2-regular clauses such that A_1 and A_2 are unifiable with most general unifier σ and A_1 and $\neg A_2$ are dominating literals in C_1 and C_2, respectively. Let all the non-constant Skolem functions in C_1 and C_2 be k_1'-originated and k_2'-originated, respectively. Then $k_1' = k_2'$ and all the non-constant Skolem functions in $(D_1 \cup D_2)\sigma$ are k_1'-originated.*

Proof. If neither C_1 nor C_2 contain function symbols, the lemma is trivially true. W.l.o.g. we assume that at least one function symbol occurs in C_1. So, the clause C_1 contains a deep literal and A_1 is deep. Therefore, there exists a compound term t_1 in A_1 such that t_1 dominates every argument of every literal in C_1. According to the possible types of $\neg A_2$, we distinguish the following cases.
1. $\neg A_2$ is a singular literal with variable x. Then $\sigma = \{x/t_1\}$. The resolvent $(D_1 \cup D_2)\sigma$ will contain only Skolem functions of $\{A_1\}$. Since all Skolem functions in $\{A_1\} \cup D_1$ are k_1'-originated, this will be the case for $(D_1 \cup D_2)\sigma$ as well.

2. $\neg A_2$ is a literal of non-singular and shallow. Similar to the previous case.
3. $\neg A_2$ is a deep literal. Then there is a term t_2 in $\neg A_2$ dominating every argument of every literal in $\{\neg A_2\} \cup D_2$. According to Lemma 2(2) the terms t_1 and t_2 occur at the same argument position in A_1 and $\neg A_2$, respectively. Since A_1 and A_2 are unifiable, t_1 and t_2 are unifiable, too. Hence, the top function symbols of t_1 and t_2 are equal. This function symbol is both k'_1-originated and k'_2-originated according to our assumptions. Thus, $k'_1 = k'_2$. Consequently, all non-constant function symbols in $(D_1 \cup D_2)\sigma$ are k'_1-originated. □

Corollary 19. *Let C_1 be an inhabited, strongly k_1-regular clause and C_2 be an inhabited, strongly k_2-regular clause such that $k_1 \neq k_2$. Then C_1 and C_2 have no $\succ_S$-resolvent.*

Thus by Lemma 18 and Corollary 19:

Lemma 20. *Let C_1 be an indecomposable, strongly k_1-regular clause and C_2 an indecomposable, strongly k_2-regular clause such that C_1 and C_2 are variable-disjoint. Let C be a $\succ_S$-resolvent of C_1 and C_2. Every split component of C is strongly k-regular for some k.*

Theorem 21. *The procedure described in Section 5 is a decision procedure for $\overline{\mathrm{DKC}}$ and $\overline{\mathrm{DK}}$.*

7 Conclusion

The results presented in this paper together with the result in [7] concerning fragments of first-order logic including transitivity based on the ordered chaining calculus, and the result in [6] concerning the guarded fragment, show that for a wide range of first-order fragments resolution-based decision procedures can be provided in a uniform framework.

We may ask whether the limits of decidability can be pushed further than $\overline{\mathrm{DK}}$. It follows from a recent result in [8] that allowing transitive relations leads to undecidability. Even though there are a number of weaker classes which, extended with equality are decidable, the enrichment of $\overline{\mathrm{K}}$ with equality gives an undecidable class [10]. We could imagine that decidable extensions of $\overline{\mathrm{K}}$ or $\overline{\mathrm{DK}}$ with restricted forms of equality can be obtained by following [5]. However, it should be kept in mind that even equalities between constants are non-trivial in our context, since their application may destroy the regularity of a literal.

The class $\overline{\mathrm{DK}}$ covers the relational translation of modal formulae of basic modal logic as well as the correspondence properties of many modal axiom schemata. Therefore, it is interesting to investigate the relation of $\overline{\mathrm{DK}}$ to the guarded fragment. From the perspective of first-order logic, the two fragments are incomparable. The formula ϕ_2 of Section 2 belongs to $\overline{\mathrm{DK}}$, but not to the guarded fragment, while it is the opposite for the formula ϕ_4. However, from a modal perspective, the class $\overline{\mathrm{DK}}$ may be regarded as a generalisation of Boolean modal logic, the multi-modal logic defined over families of binary relations closed under the Boolean operations [9].

Acknowledgements. We wish to thank the anonymous referees for their helpful comments. The work of the first author was supported by EPSRC Grant GR/K57282.

References

1. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.
2. L. Bachmair and H. Ganzinger. A theory of resolution. Research report MPI-I-97-2-005, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1997. To appear in J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning.*
3. H. de Nivelle. *Ordering Refinements of Resolution.* PhD thesis, Technische Universiteit Delft, The Netherlands, October 1996.
4. C. Fermüller, A. Leitsch, T. Tammet, and N. Zamov. *Resolution Method for the Decicion Problem*, volume 679 of *LNCS.* Springer, 1993.
5. C. Fermüller and G. Salzer. Ordered paramodulation and resolution as decision procedures. In *Proc. LPAR'93*, volume 698 of *LNAI*, pages 122–133. Springer, 1993.
6. H. Ganzinger and H. de Nivelle. A superposition decision procedure for the guarded fragment with equality. To appear in *Proc. LICS'99.* IEEE Computer Soc. Press, 1999.
7. H. Ganzinger, U. Hustadt, C. Meyer, and R. A. Schmidt. A resolution-based decision procedure for extensions of K4. To appear in *Advances in Modal Logic, Volume 2.* CLSI Publications, 1999.
8. H. Ganzinger, C. Meyer, and M. Veanes. The two-variable guarded fragment with transitive relations. To appear in *Proc. LICS'99.* IEEE Computer Soc. Press, 1999.
9. G. Gargov and S. Passy. A note on Boolean modal logic. In P. P. Petkov, editor, *Mathematical Logic: Proceedings of the 1988 Heyting Summerschool*, pages 299–309, New York, 1990. Plenum Press.
10. W. D. Goldfarb. The unsolvability of the Gödel class with identity. *Journal of Symbolic Logic*, 49:1237–1252, 1984.
11. U. Hustadt and R. A. Schmidt. Issues of decidability for description logics in the framework of resolution. To appear in R. Caferra and G. Salzer, editors, *Proc. FTP'98, LNAI.* Springer, 1999.
12. D. G. Kuehner. A note on the relation between resolution and Maslov's inverse method. In B. Meltzer and D. Michie, editors, *Machine Intelligence 6*, chapter 5, pages 73–76. Edinburgh University Press, 1971.
13. S. Ju. Maslov. The inverse method for establishing deducibility for logical calculi. In V. P. Orevkov, editor, *The Calculi of Symbolic Logic I: Proceedings of the Steklov Institute of Mathematics edited by I.G. Petrovskiĭ and S. M. Nikol'skiĭ, number 98 (1968)*, pages 25–96. American Mathematical Soc., 1971.
14. M. Mortimer. On languages with two variables. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 21:135–140, 1975.
15. C. Weidenbach. Minimal resolution. Research report MPI-I-94-227, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1994.

Prefixed Resolution
A Resolution Method for Modal and Description Logics

Carlos Areces, Hans de Nivelle, and Maarten de Rijke*

ILLC, University of Amsterdam, Plantage Muidergracht 24
1018 TV Amsterdam, The Netherlands
{carlos,nivelle,mdr}@wins.uva.nl

Abstract. We provide a resolution-based proof procedure for modal and description logics that improves on previous proposals in a number of important ways. First, it avoids translations into large undecidable logics, and works directly on modal or description logic formulas instead. Second, by using labeled formulas it avoids the complexities of earlier propositional resolution-based methods for modal logic. Third, it provides a method for manipulating so-called assertional information in the description logic setting. And fourth, we believe that it combines ideas from the method of prefixes used in tableaux and resolution in such a way that some of the heuristics and optimizations devised in either field are applicable.

1 Introduction

In this paper we develop a novel *direct* resolution method for modal logics and description logics. Designing resolution methods that can directly (that is, without having to perform translations) be applied to modal logics, received quite a bit of attention in the late 1980s and early 1990s, cf. [12,17,8]. In contrast, recent years have witnessed an increase of attention for translation-based resolution calculi for modal (and modal-like) logics; here, one translates modal languages into a large background language (typically first-order logic), and devises strategies that guarantee termination for the fragment corresponding to the original modal language; see [14,16,9].

In parallel with these developments, the description logic community has been very active in devising tableaux-based methods. There is some work on devising translation-based resolution methods for description logics [20,16], but we are not aware of any work on *direct* resolution-based methods for description logics. This is surprising for at least two reasons. First, description logics are closely related to modal logics (see [18,10]), and, hence, tools in one field can easily be used in the other. Secondly, and more importantly, in contrast with modal logic, the field of description logic has a very strong focus on decision

* Hans de Nivelle and Maarten de Rijke were supported by the Spinoza Project 'Logic in Action' at ILLC, The University of Amsterdam. We thank the referees for their careful reviews and suggestions of improvements and further work. Special thanks to Carsten Lutz for his very detailed and helpful comments.

methods and computational tools, and, surely, resolution-based methods rather than tableaux-based methods provide the basis for most of today's powerful computational logic tools, and so it seems natural to try and apply the former in the setting of description logics.

Now, *translation-based* approaches to resolution for modal and description logics suffer from the drawback that termination becomes a highly non-trivial task as we are now working within full first-order logic. Existing *direct* resolution methods for modal logics, on the other hand, lack the elegance and efficiency of the original resolution method because they need to perform 'cuts' inside modal operators to achieve completeness.

In this paper we develop a direct resolution method for modal and description logics that retains as much as possible of the lean 'one-rule' character of traditional resolution methods. The key idea introduced here is to use labels to decorate formulas with additional information. These labels encode accessibility relations (or, in description logic terms, roles) as well as worlds (or, objects). The use of labels allows us to avoid the complexities involved in previous proposals for direct resolution methods for modal logics. The intuition is that labels make information explicit as we need it, so that the basic resolution rule only needs to be used 'at the top level.'

The main achievements of this paper can be summarized as follows:

– it proposes a resolution method that does not involve skolemization beyond the use of constants;
– it presents an elegant and direct propositional resolution calculus for classical modal and description logic;
– description logics split information in two kinds: A-boxes which contain assertional information (facts about a particular domain), and T-boxes with terminological knowledge (definitions of derived notions). As far as we know, our proposal is the first one to account for assertional information with a propositional resolution approach;
– our method is hybrid and conservative in more than one sense: it allows one to adopt ideas from different fields and amalgamate them together.

The rest of the paper is organized as follows. Because of space limitations we will restrict our attention to the description logic $\mathcal{ALC}$ and its extension $\mathcal{ALCR}$; but the similarities between $\mathcal{ALC}$ and the basic multi-modal logic $\mathbf{K}_m$ are well-known [18], and they should allow anyone to transfer our results to the modal setting without problems. In Section 2 we provide some basics on description logic, and in Section 3 we present a resolution method for the description logic $\mathcal{ALCR}$. Then, in Section 4 we discuss various extensions of our results, covering both modal and description logics, and in Section 5 we point out links with related work. We conclude with a summary and further questions in Section 6.

2 Basic Issues in Description Logic

In this section we provide some background information on description logics, as well as some basic definitions.

Description logics are a family of specialized languages for the representation and structuring of knowledge, together with efficient methods to perform different 'reasoning tasks.' They are specialized languages related to the KL-ONE system of Brachman and Schmolze [6]. Nowadays, description logics are generally considered to be "variations" of first-order logic—either restrictions or restrictions plus some added operators. On the one hand these variations are motivated by the undecidability of the inference problem for first-order logic, and on the other by a desire to preserve the structure of the knowledge being represented. The main tools used for providing decision methods and studying complexity-theoretic aspects in the area of description logic are based on labeled tableaux.

Let us make things more precise now.

Definition 1 (Signature). Let $\mathcal{L} = \{C_i\} \cup \{R_i\} \cup \{a_i\}$ be a denumerable set of symbols. We will call C_i *atomic concepts*, R_i *atomic roles* and a_i *constants*. $\mathcal{L}$ is called a *signature*.

Definition 2 (Interpretation). Given a signature $\mathcal{L} = \{C_i\} \cup \{R_i\} \cup \{a_i\}$, an *interpretation* $\mathcal{I}$ for $\mathcal{L}$ is a tuple $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$, where

– Δ is a non empty set.
– $\cdot^{\mathcal{I}}$ is a function assigning an element $a_i^{\mathcal{I}} \in \Delta$ to each constant a_i; a subset $C_i^{\mathcal{I}} \subseteq \Delta$ to each atomic concept C_i; and a relation $R_i^{\mathcal{I}} \subseteq \Delta \times \Delta$ to each atomic role R_i.

Definition 3 (Concepts and Roles). Given a signature $\mathcal{L}$, each description logic will define a set of *defined concepts* and a set of *defined roles* (usually just called concepts and roles). Table 1 below defines the standard connectives together with their usual names and semantics.

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\perp$	$\emptyset$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
univ. quant.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta((d_1, d_2) \in R^{\mathcal{I}} \rightarrow d_2 \in C^{\mathcal{I}})\}$
exist. quant. ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta((d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}})\}$
role name	R	$R^{\mathcal{I}}$
role conj. ($\mathcal{R}$)	$R_1 \sqcap R_2$	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$

Table 1. Common operators of description logics

The above semantic definition of $\forall R.C$ and $\exists R.C$ matches the semantic definition of the modal operators $\Box$ and $\Diamond$; the connection was made explicit in [18].

Historically, a number of description logics received a special name; it is customary to define systems by postfixing the names of these original systems with the added operators. The logic $\mathcal{FL}^-$ [5] is defined as the description logic allowing universal quantification, conjunction and unqualified existential quantifications of the form $\exists R.\top$. The logic $\mathcal{AL}$ [19] extends $\mathcal{FL}^-$ with negation of atomic concept names. The names in parentheses in Table 1 are the usual ones for defining extensions. Hence, $\mathcal{ALC}$ is $\mathcal{AL}$ extended with full negation. In the system $\mathcal{ALCR}$ all the other operators in Table 1 can be defined.

In description logics we are interested in performing inferences given certain background knowledge.

Definition 4 (Knowledge Bases). A *knowledge base* Σ is a pair $\Sigma = \langle T, A\rangle$ such that

– T is the T(erminological)-Box, a (possibly empty) set of expressions of the forms $C_1 \sqsubseteq C_2$ or $R_1 \sqsubseteq R_2$ ($C_1, C_2 \in$ Concepts, $R_1, R_2 \in$ Roles)
– A is the A(ssertional)-Box, a (possibly empty) set of expressions of the forms $a{:}C$ or $(a, b){:}R$ ($C \in$ Concepts, $R \in$ Roles, $a, b \in$ Constants).

Definition 5 (Models). Let $\mathcal{I}$ be an interpretation and φ an expression of the kind specified below. Then $\mathcal{I}$ *models* φ (notation: $\mathcal{I} \models \varphi$) if

– $\varphi = C_1 \sqsubseteq C_2$ and $C_1^{\mathcal{I}} \subseteq {C_2}^{\mathcal{I}}$, or
– $\varphi = R_1 \sqsubseteq R_2$ and $R_1^{\mathcal{I}} \subseteq {R_2}^{\mathcal{I}}$, or
– $\varphi = a{:}C$ and $a^{\mathcal{I}} \in C^{\mathcal{I}}$, or
– $\varphi = (a, b){:}R$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

Let $\Sigma = \langle T, A\rangle$ be a knowledge base and $\mathcal{I}$ an interpretation, then $\mathcal{I}$ *models* Σ (notation: $\mathcal{I} \models \Sigma$) if for all $\varphi \in T \cup A, \mathcal{I} \models \varphi$.

Definition 6 (Reasoning Tasks). The following are some of the standard reasoning tasks considered for description logic. Let Σ be a knowledge base:

– Subsumption ($\Sigma \models C_1 \sqsubseteq C_2$): check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
– Instance Checking ($\Sigma \models a : C$): check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$.
– Concept Consistency ($\Sigma \not\models C \doteq \bot$): check whether for some interpretation $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $C^{\mathcal{I}} \neq \{\}$.
– Knowledge Base Consistency ($\Sigma \not\models \bot$): check whether there exists $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$.

Similar tasks can obviously also be defined for roles whenever role definitions have a richer structure than we have considered here.

In this paper we will be concerned with knowledge base consistency, which, in sufficiently strong description logics like $\mathcal{ALC}$ and its extensions, can decide all the other reasoning tasks.

3 Decision Methods for Description Logics

Weak logics like $\mathcal{FL}^-$ have very effective decision methods. For some of the standard reasoning tasks mentioned in Definition 6 these methods are polynomial and only need to perform a structural analysis of the concepts involved (i.e., no "real deduction" is performed). It is interesting to note that at this (low) level of expressive power the different reasoning tasks cannot be mapped into each other. But, of course, once we have full boolean expressive power as in $\mathcal{ALC}$, reasoning tasks like subsumption can be translated into satisfaction queries.

However, even at the level of $\mathcal{ALC}$ there is another dimension which matters: the difference between dealing with assertional information and terminological information. More precisely, assertions increase the expressive power of description logics. The standard connection between description logics and K_m is established at the terminological level. To account for the assertional information the notion of *nominal* or name for a world is needed. See [4] for a recent study of this topic and Section 5 for further comments.

Below we give a resolution proof procedure for the description logic $\mathcal{ALCR}$ that is able to cope with assertional information.

Definition 7 (Weak Negation Form). Define the following rewriting procedure WNF on concepts:

1. $\neg\neg C \overset{\text{WNF}}{\rightsquigarrow} C$
2. $\exists R.C \overset{\text{WNF}}{\rightsquigarrow} \neg(\forall R.\neg C)$
3. $(C_1 \sqcup \cdots \sqcup C_j) \overset{\text{WNF}}{\rightsquigarrow} \neg(\neg C_1 \sqcap \cdots \sqcap \neg C_j)$
4. $\top \overset{\text{WNF}}{\rightsquigarrow} \neg(C \sqcap \neg C)$, for C arbitrary
5. $\bot \overset{\text{WNF}}{\rightsquigarrow} \neg\top$

For any concept C, WNF converges to a unique normal form which we denote as WNF(C). WNF(C) is logically equivalent to C. WNF can trivially be extended to expressions $a : C_1$ by setting WNF$(a : C_1) = a :$ WNF(C_1). If we interpret $\sqcup$, $\exists R.C$, $\top$ and $\bot$ as defined operators, then WNF is slightly more than an expansion of definitions.

Definition 8 (Clause). Given an infinite set of labels L disjoint from Constants, a *clause* is a set Cl such that each element of Cl is either

– a concept assertion of the form $t{:}C$ where t is either a constant or a label in L.
– a role assertion of the form $(t_1, t_2) : R$, where t_1, t_2 are either constants or labels in L.

We will use the notation $t{:}C$ (with possible subindices) for concept assertions and $(t_1, t_2){:}R$ (with subindices) for role assertions, and $\bar{t}{:}N$ for both of them.

A formula in a clause is a *literal* if it is either a role assertion, a concept or negated concept assertion on an atomic concept, or a universal or negated universal concept assertion.

Definition 9 (Model for a Clause and a Set of Clauses). Notice that formulas in a clause are simply assertions over an expanded set of constants. Let Cl be a clause, and $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ a model in the expanded signature; we put $\mathcal{I} \models Cl$ if $\mathcal{I} \models \bigvee Cl$. A set of clauses S *has a model* if there is model $\mathcal{I}$ such that for all $Cl \in S$, $\mathcal{I} \models Cl$.

Definition 10 (Set of Clauses of a Knowledge Base). Let $\Sigma = \langle T, A \rangle$ be a knowledge base (with non-cyclic definitions). It is known that Σ can be transformed into an "unfolded" equivalent knowledge base $\Sigma' = \langle \emptyset, A' \rangle$ where all concept and role assertions use only atomic concept and role symbols [11].

The set S_Σ of clauses corresponding to Σ is the smallest set such that

- if $a\!:\!C_1 \sqcap \cdots \sqcap C_n = \text{WNF}(a\!:\!C)$ $(n \geq 1)$ for $a\!:\!C \in A'$ then $\{a\!:\!C_i\} \in S_\Sigma$, for $1 \leq i \leq n$.
- if $(a,b)\!:\!R_1 \sqcap \cdots \sqcap R_n \in A'$ then $\{(a,b)\!:\!R_i\} \in S_\Sigma$, for $1 \leq i \leq n$.

Notice that the "unfolded" assertions of A' are used in this translation. Furthermore, in S_Σ we can identify a (possibly empty) subset of clauses RA of the form $\{(a,b) : R\}$ which we call *role assertions*, and for each constant a a (possibly empty) subset CA_a of clauses of the form $\{a : C\}$ which we call *concept assertions.*

Because of the format of a knowledge base it is impossible to find in S_Σ *mixed* clauses containing both (in disjunction) concept and role assertions. Furthermore there are no disjunctive concept assertions on different constants, i.e., there is no clause Cl in S_Σ such that $Cl = Cl' \cup \{a : C_1\} \cup \{b : C_2\}$ for $a \neq b$. These properties will be relevant in the first steps of the completeness proof.

Proposition 1. *Let Σ be a knowledge base and S_Σ its corresponding set of clauses. Then Σ is satisfiable iff S_Σ is satisfiable.*

$$(\sqcap)\ \frac{Cl \cup \{\bar{t}\!:\!N_1 \sqcap N_2\}}{\begin{array}{c}Cl \cup \{\bar{t}\!:\!N_1\}\\ Cl \cup \{\bar{t}\!:\!N_2\}\end{array}} \qquad (\neg\sqcap)\ \frac{Cl \cup \{t\!:\!\neg(C_1 \sqcap C_2)\}}{Cl \cup \{t\!:\!\text{WNF}(\neg C_1), t\!:\!\text{WNF}(\neg C_2)\}}$$

$$(\text{RES})\quad \frac{Cl_1 \cup \{\bar{t}\!:\!N\} \quad Cl_2 \cup \{\bar{t}\!:\!\neg N\}}{Cl_1 \cup Cl_2}$$

$$(\forall)\quad \frac{Cl_1 \cup \{t_1\!:\!\forall R.C\} \quad Cl_2 \cup \{(t_1,t_2)\!:\!R\}}{Cl_1 \cup Cl_2 \cup \{t_2\!:\!C\}}$$

$$(\neg\forall)\quad \frac{Cl \cup \{t\!:\!\neg\forall R.C\}}{\begin{array}{c}Cl \cup \{(t,n)\!:\!R\}\\ Cl \cup \{n\!:\!\text{WNF}(\neg C)\}\end{array}},\ \text{where } n \text{ is new.}$$

(notice that $(\sqcap)$ also covers role conjunction and that $(\neg\forall)$ is a mild kind of skolemnization which only involves the introduction of constants).

Table 2: The Resolution Rules

Table 2 shows the resolution rules we will consider.

Definition 11 (Deduction). A *deduction* of a clause Cl from a set of clauses S is a finite sequence $S_1, \ldots, S_n$ of sets of clauses such that $S = S_1$, $Cl \in S_n$ and each S_i (for $i > 1$) is obtained from S_{i-1} by adding the consequent clauses of the application of one of the resolution rules in Table 2 to clauses in S_{i-1}. Cl is a *consequence* of S if there is a deduction of Cl from S. A deduction of $\{\}$ from S is a *refutation* of S.

Before proving soundness, completeness and termination we present a simple example of resolution in our system.

Example 1. Consider the following description. Ignoring some fundamental genetic laws, suppose that children of tall people are blond (1). Furthermore, all Tom's daughters are tall (2), but he has a non-blond grandchild (3). Can we infer that Tom has a son (4)?

$$\begin{array}{ll}
(0) & \mathsf{female} \doteq \neg\mathsf{male} \\
(1) & \mathsf{tall} \sqsubseteq \forall\mathsf{CHILD}.\mathsf{blond} \\
(2) & \mathsf{t}:\forall\mathsf{CHILD}.(\neg\mathsf{female} \sqcup \mathsf{tall}) \\
(3) & \mathsf{t}:\exists\mathsf{CHILD}.\exists\mathsf{CHILD}.\neg\mathsf{blond} \\
\hline
(4) & \mathsf{t}:\exists\mathsf{CHILD}.\mathsf{male}
\end{array}$$

As is standard, we use a new proposition letter rest-tall to complete the partial definition in (1) and we resolve with the negation of the formula we want to infer. After unfolding and applying WNF we obtain the following three clauses

1. $\{\mathsf{t}:\forall\mathsf{CHILD}.\neg(\neg\mathsf{male} \sqcap \neg((\forall\mathsf{CHILD}.\mathsf{blond}) \sqcap \mathsf{rest\text{-}tall}))\}$
2. $\{\mathsf{t}:\neg\forall\mathsf{CHILD}.\forall\mathsf{CHILD}.\mathsf{blond}\}$
3. $\{\mathsf{t}:\forall\mathsf{CHILD}.\neg\mathsf{male}\}$

Now we start resolving,

4. $\{\mathsf{t}':\neg\forall\mathsf{CHILD}.\mathsf{blond}\}$ by $(\neg\forall)$ in 2.
5. $\{(\mathsf{t},\mathsf{t}'):\mathsf{CHILD}\}$ by $(\neg\forall)$ in 2.
6. $\{\mathsf{t}':\neg\mathsf{male}\}$ by $(\forall)$ in 3.
7. $\{\mathsf{t}':\neg(\neg\mathsf{male} \sqcap \neg((\forall\mathsf{CHILD}.\mathsf{blond}) \sqcap \mathsf{rest\text{-}tall})\}$ by $(\forall)$ in 1.
8. $\{\mathsf{t}':\mathsf{male}, \mathsf{t}':((\forall\mathsf{CHILD}.\mathsf{blond}) \sqcap \mathsf{rest\text{-}tall})\}$ by $(\neg\sqcap)$ in 7.
9. $\{\mathsf{t}':((\forall\mathsf{CHILD}.\mathsf{blond}) \sqcap \mathsf{rest\text{-}tall})\}$ by (RES) in 6 and 8.
10. $\{\mathsf{t}':\forall\mathsf{CHILD}.\mathsf{blond}\}$ by $(\sqcap)$ in 9.
11 $\{\mathsf{t}':\mathsf{rest\text{-}tall}\}$. by $(\sqcap)$ in 9.
12. $\{\}$ by (RES) in 4 and 10.

Theorem 1 (Soundness). *The resolution rules described in Table 2 are sound. That is, if Σ is a knowledge base, then S_Σ has a refutation only if Σ is unsatisfiable.*

Proof. We will prove that the resolution rules we introduced preserve satisfiability. That is, given a rule, if the premises are satisfiable, then so are the conclusions. We only discuss $(\neg\forall)$.

Let $\mathcal{I}$ be a model of the antecedent. If $\mathcal{I}$ is a model of Cl we are done. If $\mathcal{I}$ is a model of $t:\neg\forall R.C$, then there exists d in the domain, such that $(t^{\mathcal{I}}, d) \in R^{\mathcal{I}}$ and $d \in \neg C^{\mathcal{I}}$. Let $\mathcal{I}'$ be identical to $\mathcal{I}$ except perhaps in the interpretation of n where $n^{\mathcal{I}'} = d$. As n is a new label, also $\mathcal{I}' \models t:\neg\forall R.C$. But now $\mathcal{I}' \models Cl \cup \{(t,n):R\}$ and $\mathcal{I}' \models Cl \cup \{n:\mathrm{WNF}(\neg C)\}$. $\square$

Our next aim is to prove completeness. We follow the approach used in [12]: given a set of clauses S we aim to define a structure T_S such that

(†) if S is satisfiable, a model can be effectively constructed from T_S; and
(††) if S is unsatisfiable, a refutation can be effectively constructed from T_S.

But in our case this proves to be more difficult than in [12] because we have to deal with A-Box information, that is, with named objects or worlds (concept assertions) and fixed constraints on relations (role assertions). We will proceed in stages. To begin, we will obtain a first structure to account for named worlds and their fixed relation constraints. After that we can use a simple generalization of results in [12]. We base our construction on trees which will help in guiding the construction of the corresponding refutation proof.

Let Σ be a knowledge base and S_Σ its corresponding set of clauses. Let a be a constant and CA_a the subset of CA of concept assertions concerning the constant a. Define the following operation to be performed on CA_a.

We construct for each CA_a a binary tree T_a inductively. Let the original tree u consist of the single node CA_a and repeat the following operations in an alternating fashion.

Operation A1. Repeat the following steps as long as possible:
- choose a leaf w. Replace any clause of the form $\{a : \neg(C_1 \sqcap C_2)\}$ by $\{a : \mathrm{WNF}(\neg C_1), a : \mathrm{WNF}(\neg C_2)\}$; and any clause of the form $\{a : C_1 \sqcap C_2\}$ by $\{a{:}C_1\}$ and $\{a{:}C_2\}$.

Operation A2. Repeat the following steps as long as possible:
- choose a leaf w of u and a clause Cl in w of the form $Cl = \{a{:}C_1, a{:}C_2\} \cup Cl'$;
- add two children w_1 and w_2 to w, where $w_1 = w\backslash\{Cl\} \cup \{\{a : C_1\}\}$ and $w_2 = w\backslash\{Cl\} \cup \{\{a{:}C_2\} \cup Cl'\}$.

The leaves of T_a give us the possibilities for "named worlds" in our model (remember that concept prefixes act as names for worlds/objects). We can view each leaf as a set S_a^j, representing a possible configuration for world a.

Proposition 2. *Operation A (the combination of A1 and A2) terminates, and upon termination*

1. *all the leaves S_a^1 to S_a^n of the tree are sets of unit literal clauses;*
2. *if all $S_a^1, \ldots, S_a^n$ are refutable, then CA_a is refutable;*
3. *if one S_a^j is satisfiable, then CA_a is satisfiable.*

Proof. Termination is trivial. Item 1 holds by virtue of the construction, and item 2 is proved by induction on the depth of the tree. We need only realize that by simple propositional resolution if the two children of a node w are refutable, then so is w. Item 3 is also easy. Informally, Operation A "splits" disjunctions and "carries along" conjunctions. Hence if some S_a^j has a model we have a model satisfying all conjuncts in CA_a and at least one of each disjuncts. □

We should now consider the set RA of role assertions. Let NAMES be the set of constants which appear in Σ. If a is in NAMES but CA_a is empty in S_Σ, define $S_a^1 = \{\{a : C, a : \neg C\}\}$ for some concept C. We will construct a set of sets of nodes $\mathcal{N} = \{N_i \mid N_i$ contains exactly one leaf of each $T_a\}$. Each N_i is a possible set of constraints for the named worlds in a model of S_Σ.

Proposition 3. *If for all i, $\bigcup N_i \cup RA$ is refutable, then so is S_Σ.*

Proof. If for all i, $\bigcup N_i \cup RA$ is refutable, then for some constant a we have that for all S_a^j obtained from CA_a, $S_a^j \cup RA$ is refutable. Hence by Proposition 2, $CA_a \cup RA$ is refutable, and so is S_Σ. □

For all i, we will now extend each set in N_i with further constraints. For each $S_a \in N_i$, start with a node w_a labeled by S_a .

Operation B1. Equal to *Operation A1.*

Operation B2. Repeat the following steps as long as possible:
- choose nodes w_a, w_b such that $\{(a,b){:}R\}$ in RA, $\{a{:}\forall R_i.C_i\} \in w_a$, $\{b{:}C_i\} \notin w_b$, where w_b is without children;
- add a child to w_b, $w_b' = w_b \cup \{\{b{:}C_i\}\}$.

Call N_i^* the set of all leaves obtained from the forest constructed in B.

Proposition 4. *Operation B terminates, and upon termination*

1. *all nodes created are derivable from $\bigcup N_i \cup RA$, and hence if a leaf is refutable so is $\bigcup N_i \cup RA$;*
2. *if some $\bigcup N_i^*$ is satisfiable, then S_Σ is satisfiable.*

Proof. To prove termination, notice that in each cycle the quantifier depth of the formulas considered decreases. Furthermore, it is not possible to apply twice the operation to a node named by a and b and a formula $a{:}\forall R_i.C_i$.

As to item 1, each node is created by an application of the $(\forall)$ rule to members of $N_i \cup RA$ or clauses previously derived by such applications. To prove item 2, let $\mathcal{I}$ be a model of N_i^*. Define a new model $\mathcal{I}' = \langle \Delta', \cdot^{\mathcal{I}'} \rangle$ as follows.

- $\Delta' = \Delta$;
- $a^{\mathcal{I}'} = a^{\mathcal{I}}$ for all constants a;
- $C^{\mathcal{I}'} = C^{\mathcal{I}}$ for all atomic concepts C; and
- $R^{\mathcal{I}'} = R^{\mathcal{I}} \cup \{(a^{\mathcal{I}}, b^{\mathcal{I}}) \mid \{(a,b){:}R\} \in RA\}$.

Observe that $\mathcal{I}'$ differs from $\mathcal{I}$ only in an extended interpretation of role symbols. By definition, $\mathcal{I}' \models RA$. It remains to prove that $\mathcal{I}' \models CA$. By Proposition 2, we are done if we prove that $\mathcal{I}' \models \bigcup N_i^*$. Now, since we only expanded the interpretation of relations, $\mathcal{I}$ and $\mathcal{I}'$ can only disagree on universal concepts of the form $a{:}\forall R.C$. By induction on the quantifier depth we prove this to be false.

Assume that $\mathcal{I}$ and $\mathcal{I}'$ agree on all formulas of quantifier depth less than n, and let $a : \forall R.C$ be of quantifier depth n, for $\{a : \forall R.C\} \in S_a^*$. Suppose $\mathcal{I}' \not\models \forall R.C$. This holds iff there exists b such that $(a^{\mathcal{I}'}, b^{\mathcal{I}'}) \in R^{\mathcal{I}'}$ and $\mathcal{I}' \not\models b{:}C$. By the inductive hypothesis, $\mathcal{I} \not\models b : C$. Now, if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ we are done. Otherwise, by definition $\{(a,b){:}R\} \in RA$. But then $\{b{:}C\} \in S_b^*$ by construction and as $\mathcal{I} \models S_b^*$, we also have $\mathcal{I} \models b{:}C$—a contradiction. □

As we said above, each N_i^* represents the "named core" of a model of S. The final step is to define the non-named part of the model. The following operations are performed to each set in each of the N_i^* obtaining in such a way a forest F_i.

Fix N_i^*, and a. We construct a tree "hanging" from the corresponding $S_a^* \in N_i^*$. The condition that each node of the tree is named by either a constant of a new label (that is, all the formulas have the same prefix) will be preserved as an invariant during the construction. Set the original tree u to S_a^* and repeat the following operations C1, C2 and C3 in succession until the end-condition holds.

Operation C1. Equal to *Operation A1.*

Operation C2. Equal to *Operation A2.*

Operation C3. For each leaf w of u,

- if for some concept we have $\{C\}, \{\neg C\} \in w$, do nothing;
- otherwise, since w is a set of unit clauses, we can write $w = \{\{t : C_1\}, \ldots, \{t : C_m\}, \{t : \forall R_{k_1}.A_1\}, \ldots, \{t : \forall R_{k_n}.A_n\}, \{t : \neg\forall R_{l_1}.P_1\}, \ldots, \{t : \neg\forall R_{l_q}.P_q\}\}$. Form the sets $w_i = \{\{\mathrm{WNF}(t' : \neg P_i)\}\} \cup S_i$, where t' is a new label, and $S_i = \{\{t' : A_h\} \mid \{t : \forall R_i.A_h\} \in w\}$, and append each of them to w as children marking the edges as R_i links. The nodes w_i are called the *projections* of w.

End-condition. Operation C3 is inapplicable.

Proposition 5. *Operation C cannot be applied indefinitely.*

Definition 12. We call nodes to which Operation C1 or C2 has been applied of type 1, and those to which Operation C3 has been applied of type 2. The set of *closed nodes* is recursively defined as follows,

- if for some concept $\{t : C\}, \{t : \neg C\}$ are in w then w is closed.
- if w is of type 1 and all its children are closed, w is closed.
- if w is of type 2 and some of its children is closed, w is closed.

Let F_i be a forest that is obtained by applying Operations C1, C2, and C3 to N_i^* as often as possible. Then F_i is *closed* if any of its roots is closed.

Lemma 1. *If one of the forest F_i obtained from S_Σ is non-closed, then S_Σ has a model.*

Proof. Let F_i be a non-closed forest. By a simple generalization of the results in [12, Lemma 2.7] we can obtain a model $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ of all roots S_a^* in F_i, from the trees "hanging" from them, ie., a model of $\bigcup N_i^*$. By Proposition 4, S_Σ has a model. □

Lemma 1 establishes the property (†) we wanted in our structure T_S. To establish (††) we need a further auxiliary result.

Proposition 6. *Let w be a node of type 2. If one of its projections w_i is refutable, then so is w.*

Proof. Let w be a set of unit clauses $w = \{\{t\!:\!C_1\}, \ldots, \{t\!:\!C_m\}, \{t\!:\!\forall R_{k_1}.A_1\}, \ldots, \{t\!:\!\forall R_{k_n}.A_n\}, \{t\!:\!\neg\forall R_{l_1}.P_1\}, \ldots, \{t\!:\!\neg\forall R_{l_q}.P_q\}\}$. And let w_i be its refutable projection: $w_i = \{\{\mathrm{WNF}(t'\!:\!\neg P_i)\}\} \cup S_i$, where t' is a new label, and $S_i = \{\{t'\!:\!A_h\} \mid \{t\!:\!\forall R_i.A_h\} \in w\}$. We use resolution on w to arrive to the clauses in w_i from which the refutation can carried out: Apply $(\neg\forall)$ to $\{t\!:\!\neg\forall R_i.P_i\}$ in w to obtain $\{t' : \mathrm{WNF}(t' : \neg P_i)\}$ and $\{(t, t') : R_i\}$. Now apply $(\forall)$ to all the clauses $\{t\!:\!\forall R_i.A_h\}$ in w to obtain $\{t'\!:\!A_h\}$. □

Lemma 2. *In a forest F_i, every closed node is refutable.*

Proof. For w a node in F_i, let $d(w)$ be the longest distance from w to a leaf.

If $d(w) = 0$, then w is a leaf, thus for some concept C, $\{t\!:\!C\}$ and $\{t\!:\!\neg C\}$ are in w. Using (RES) we immediately derive $\{\}$.

For the induction step, suppose the proposition holds for all w' such that $d(w') < n$ and that $d(w) = n$. If w is of type 1, let $w_1 = w \setminus \{Cl\} \cup \{Cl_1\}$ and $w_2 = w \setminus \{Cl\} \cup \{Cl_2\}$ be its children. By the inductive hypothesis there is a refutation for w_1 and w_2. By propositional resolution there is a refutation of w: repeat the refutation proof for w_2 but starting with w, instead of the empty clause we should obtain a derivation of Cl_2, now use the refutation of w_2.

Suppose w is of type 2. Because w is closed, one of its projections is closed. Hence, by the inductive hypothesis it has a refutation. By Proposition 6, w itself has a refutation. □

Theorem 2 (Completeness). *The resolution method described above is complete: if Σ is a knowledge base, then S_Σ is refutable whenever Σ is unsatisfiable.*

Proof. It is only necessary to put together the previous pieces. If Σ does not have a model then, by Proposition 1, there is no model for S_Σ. Hence by Lemma 1 all the forests F_i obtained from S_Σ are closed, and by Lemma 2, for each N_i^*, one of the sets $S_{a_j}^*$ is refutable. By Proposition 4, for all i, $\bigcup N_i \cup RA$ is refutable. By Proposition 3, S_Σ is refutable. □

Because we have shown how to *effectively* obtain a refutation from an inconsistent set of clauses we have also established termination. Notice that during the completeness proof we have used a *specific strategy* in the application of the resolution rules (for example, the $(\neg\forall)$ rule is never applied twice to the same formula).

Theorem 3 (Termination). *Given a knowledge base Σ, the resolution method (with the strategy described above) terminates with answer **YES** if Σ is inconsistent and with answer **NO** otherwise.*

As a corollary of the results above, we obtain soundness, completeness and termination of our resolution method for $\mathbf{K}_m$. Notice that this is really a weaker result than the ones proved above, since we don't have to bother about assertional A-Box information when dealing with $\mathbf{K}_m$. When using our resolution method for $\mathbf{K}_m$ the prefix labels are really metalogical entities and not part of the logic. We will discuss this matter further in Section 5.

4 Extensions and Variations

In addition to the basic results in Section 3, we will now discuss some extensions and variations. Because of space constraints we provide few details.

Modal Extensions. The natural step, from a classical modal point of view, is to consider systems above $\mathbf{K}_m$. We choose systems **T**, **D**, and **4** as examples. Each system is defined as an extension of the basic system **K** by the addition of an axiom scheme which characterizes certain properties of the accessibility relation:

Name	Axiom scheme	Accessibility Relation
T	$p \to \Diamond p$	reflexivity: $\forall x.xRx$
D	$\Box p \to \Diamond p$	seriality: $\forall x \exists y.xRy$
4	$\Diamond\Diamond p \to \Diamond p$	transitivity: $\forall xyz.(xRy \wedge yRz \to xRz)$

Corresponding to each of the axioms we add a new resolution rule.

$$(\mathbf{T}_i) \quad \frac{Cl \cup \{t{:}\forall R_i.C\}}{Cl \cup \{t{:}C\}}$$

$$(\mathbf{D}_i) \quad \frac{Cl \cup \{t{:}\forall R_i.C\}}{Cl \cup \{t{:}\neg\forall R_i.\mathrm{WNF}(\neg C)\}}$$

$$(\mathbf{4}_i) \quad \frac{Cl_1 \cup \{t_1{:}\forall R_i.C\} \quad Cl_2 \cup \{(t_1,t_2){:}R_i\}}{Cl_1 \cup Cl_2 \cup \{t_2 : \forall R_i.C\}}$$

Of course, because we are in a multi-modal formalism, these rules can be specified for any particular relation R_i. From the description logic point of view these extensions can be understood as forcing certain properties on a specific relation. There exist description logics which permit the definition of the reflexive-transitive closure of a relation (R^*). Seriality is related to functionality of roles, another feature common in description logic formalisms.

Soundness for these systems is immediate:

Theorem 4. *The resolution methods obtained by adding the rules* (**T**), (**D**) *and* (**4**) *for a particular relation R_i, are sound with respect to the class of knowledge bases where the relation R_i is always interpreted as reflexive, serial and transitive, respectively.*

For completeness and termination we should modify the construction we defined previously (in particular ($\mathbf{4}_i$) needs a mechanism of cycle detection); this can be done again using methods from [12].

Theorem 5. *The resolution methods obtained by adding the rules* (**T**), (**D**) *and* (**4**) *for a particular relation R_i, are complete and terminate with respect to the class of knowledge bases where the relation R_i is always interpreted as reflexive, serial and transitive, respectively.*

DL Extensions. In the description logic community one considers a kind of extensions of the language that is different from the ones we already introduced. For instance, recently in [7] some attention has been given to n-ary relations in description logics (in modal logic terms, n-dimensional modal operators). Our approach seems to generalize without further problems to account for this.

Finally, another direction for extensions is to consider additional structure on roles. We have limited ourselves to conjunction, but disjunction, negation, composition, etc. can be considered. Description logics allowing these operations are known as *very expressive description logics*, and their worst case complexity is high, even though they perform well in some limited cases; their modal logic counterparts are related to dynamic logics based on PDL. For a translation based resolution treatment of these, see [16].

5 Related Work

The Connection with Resolution for Modal Logics. Resolution methods for modal logics (without translation) have been investigated before [13,17,12,8]. The innovation introduced in this paper is in the use of labels. We think this is the key to simplify the complexities involved in previous proposals. Previously, resolution had to be performed "inside" modalities (in a similar way as how new tableaux had to be started in non-prefixed tableaux systems). Labels allow us to make information explicit and resolution can then always be performed at the "top level." Because we have labels available, we can also deal with properties on relations—like reflexivity, seriality or transitivity—in a tableaux-like fashion, and a single new rule is all that is necessary to account for them.

Comparison with the Tableaux Method. Once labels are introduced the resolution method is very close to the tableaux approach, but we are still doing resolution. The rules ($\sqcap$), ($\neg\sqcap$) and ($\neg\forall$), prepare formulas to be "fed" into the resolution rules (RES) and ($\forall$).[1] And the aim is still to derive the empty clause instead of finding a model by exhausting a branch.

But, is this method any better than tableaux? We don't think this is the correct question to ask. We believe that we learn different things from studying different methods. For example, [15] studies a number of interesting optimizations of the tableaux implementation which were tested on the tableaux based theorem prover DLP. Some of their ideas were already incorporated in our resolution method (lexical normalization and early detection of clashes), and others might perhaps be used in implementations of our method. But what is perhaps more interesting to the description logic community, is that new optimizations, specific to the resolution approach, can now be exploited.

Strategies for Modal Resolution. In implementations of the resolution algorithm, strategies for selecting the resolving pairs are critical. Heuristics for the case of modal logics have been investigated in [1]. Some of their results extend to our framework, and in certain cases proofs are simpler because of our explicit use of resolution via labels. We cannot give a full description of this issue here.

[1] ($\forall$) is added to account for the "hidden" negation in the guard of the quantifier.

Assertional Information and Hybrid Logics. There is a final topic on which we would like to comment: the relation between nominals and assertional information. The similarity between $\mathcal{ALC}$ and the basic multi-modal logic $\mathbf{K}_m$ is well-known. But this connection concerns the terminological part of $\mathcal{ALC}$. Recent work on nominals and hybrid languages [4] explains how assertions enter the picture. This paper investigates $\mathcal{ALCNO}$, $\mathcal{ALC}$ plus counting, plus the set formation operator in terms of individuals: $\mathcal{O}(a_1, \ldots, a_n)^{\mathcal{I}} = \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$, embedding this logic in a very expressive hybrid formalism $\mathcal{H}(\forall)$. To account for $\mathcal{ALC}$ (including assertions) a subset of a weaker system called $\mathcal{H}(@)$ is enough. For this language, labeled tableaux appear as a very natural choice [3]; however, by using our labeled calculus, resolution has become just as natural a choice.

6 Conclusions and Further Work

In this paper we have provided a propositional resolution method for deciding knowledge base consistency for $\mathcal{ALCR}$. This result is further extended to account for reflexive, serial and transitive relations. Because of the connection between $\mathcal{ALC}$ and $\mathbf{K}_m$, our methods can also be used as resolution methods for deciding theoremhood of modal logics. Due to space limitations, only the basics of related issues such as more expressive description logics, and optimized strategies where discussed. These issues are being dealt with in the full version of the paper.

There is a number of important questions which are still open at this stage of our research. First, up to now we have no implementation, but this issue is high on our agenda. We believe that the ideas behind our resolution method are simple enough so that even adapting already available provers should not prove to be a very difficult task.

Further, a very attractive idea which matches nicely with the resolution approach is to incorporate a limited kind of subsumption on universal prefixes to account for "on the fly" unfolding of terminological definitions. The use of such "universal labels" should make it unnecessary to perform a complete unfolding of the knowledge base as a pre-processing step: The leitmotiv would be "perform expansion by definitions only when needed in deduction." On the fly unfolding has already been implemented in tableaux based systems like KRIS [2].

As to the complexity of resolution: we have not attempted to formally establish the complexity of our resolution method so far. We conjecture that a PSPACE heuristic for prefixed resolution exists, even though in this first account the naïve heuristic we have introduced requires exponential space.

Finally, our completeness proof is constructive: when a refutation cannot be found we can actually define a model for the knowledge base. Hence, our method can also be used for model extraction. How does this method perform in comparison with traditional model extraction from tableaux systems?

References

1. Y. Auffray, P. Enjalbert, and J. Hebrard. Strategies for modal resolution: results and problems. *Journal of Automated Reasoning*, 6(1):1–38, 1990.
2. F. Baader, B. Hollunder, B. Nebel, H. Profitlich, and E. Franconi. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Journal of Applied Intelligence*, 4:109–132, 1994.
3. P. Blackburn. Internalizing labeled deduction. Technical Report CLAUS-Report 102, Computerlinguistik, Universität des Saarlandes, 1998.
4. P. Blackburn and M. Tzakova. Hybridizing concept languages. *Annals of Mathematics and Artificial Intelligence*, 24:23–49, 1998.
5. R. Brachman and H. Levesque. The tractability of subsumption in frame-based description languages. In *AAAI-84*, pages 34–37, 1984.
6. R. Brachman and J. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
7. D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive query containment in description logics with n-ary relations. In *Proc. of DL-97*, pages 5–9, June 1997.
8. H. de Nivelle. *Ordering refinements of resolution.* PhD thesis, Technische Universiteit Delft, Delft. The Netherlands, October 1994.
9. H. de Nivelle and M. de Rijke. Resolution decides the guarded fragment. Submitted for journal publication, 1998.
10. M. de Rijke. Restricted description languages. Unpublished, 1998.
11. F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Deduction in concept languages: from subsumption to instance checking. *Journal of Logic and Computation*, 4(4):423–452, 1994.
12. P. Enjalbert and L. Fariñas del Cerro. Modal resolution in clausal form. *Theoretical Computer Science*, 65(1):1–33, 1989.
13. L. Fariñas del Cerro. A simple deduction method for modal logic. *Information Processing Letters*, 14:49–51, April 1982.
14. C. Fermüller, A. Leitsch, T. Tammet, and N. Zamov. *Resolution methods for the decision problem.* Springer-Verlag, Berlin, 1993. LNAI.
15. I. Horrocks and P. Patel-Schneider. Optimising description logic subsumption. Submitted to the Journal of Logic and Computation, 1998.
16. U. Hustadt and R. Schmidt. Issues of decidability for description logics. Unpublished, 1998.
17. G. Mints. Resolution calculi for modal logics. *American Mathematical Society Translations*, 143:1–14, 1989.
18. K. Schild. A correspondence theory for terminological logics. In *Proc. of the 12th IJCAI*, pages 466–471, 1991.
19. M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.
20. T. Tammet. Using resolution for extending KL-ONE-type languages. In *Proc. of the 4th CIKM.* ACM Press, November 1995.

System Description: Twelf — A Meta-Logical Framework for Deductive Systems

Frank Pfenning and Carsten Schürmann[⋆]

Department of Computer Science
Carnegie Mellon University
fp@cs.cmu.edu, carsten@cs.cmu.edu

Abstract. Twelf is a meta-logical framework for the specification, implementation, and meta-theory of deductive systems from the theory of programming languages and logics. It relies on the LF type theory and the judgments-as-types methodology for specification [HHP93], a constraint logic programming interpreter for implementation [Pfe91], and the meta-logic $\mathcal{M}_2$ for reasoning about object languages encoded in LF [SP98]. It is a significant extension and complete reimplementation of the Elf system [Pfe94].
Twelf is written in Standard ML and runs under SML of New Jersey and MLWorks on Unix and Window platforms. The current version (1.2) is distributed with a complete manual, example suites, a tutorial in the form of on-line lecture notes [Pfe], and an Emacs interface. Source and binary distributions are accessible via the Twelf home page `http://www.cs.cmu.edu/~twelf`.

1 The Twelf System

The Twelf system is a tool for experimentation in the theory of programming languages and logics. It supports a variety of tasks which we explain in this section: *specification* of object languages and their semantics, implementation of *algorithms* manipulating object-language expressions and deductions, and formal development of the *meta-theory* of an object language. Several extensive experiments have been conducted with Twelf, such as the formal development of the theory of logic and functional programming languages and various logics.

Specification. Twelf employs the representation methodology and underlying type theory of the LF logical framework. Expressions are represented as LF objects using the technique of *higher-order abstract syntax* whereby variables of an object language are mapped to variables in the meta-language. This means that common operations, such as renaming of bound variables or capture-avoiding substitution are directly supported by the framework and do not need to be programmed anew for each object language.

[⋆] This work was supported by NSF Grant CCR-9619584.

For semantic specification LF uses the *judgments-as-types* representation technique. This means that a derivation is coded as an object whose type represents the judgment it establishes. Checking the correctness of a derivation is thereby reduced to type-checking its representation in the logical framework (which is efficiently decidable).

Algorithms. Generally, specification is followed by implementation of algorithms manipulating expressions or derivations. Twelf supports the implementation of such algorithms by a constraint logic programming interpretation of LF signatures, a slight variant of the one originally proposed in [Pfe91] and implemented in Elf [Pfe94]. The operational semantics is based on goal-directed, backtracking search for an object of a given type.

Meta-Theory. Twelf provides two related means to express the meta-theory of deductive systems: higher-level judgments and the meta-logic $\mathcal{M}_2$.

A higher-level judgment describes a relation between derivations inherent in a (constructive) meta-theoretic proof. Using the operational semantics for LF signatures sketched above, we can then execute a meta-theoretic proof. While this method is very general and has been used in many of the experiments mentioned below, type-checking a higher-level judgment does not by itself guarantee that it correctly implements a proof.

Alternatively, one can use an experimental automatic meta-theorem proving component based on the meta-logic $\mathcal{M}_2$ for LF [SP98]. It expects as input a Π_2 statement about closed LF objects over a fixed signature and a termination ordering and searches for an inductive proof. If one is found, its representation as a higher-level judgment is generated and can then be executed.

Even though a number of the theorems in the example suites described below can be proven automatically, we consider the meta-theorem prover to be in a preliminary state. Its main current limitations are the lack of automatic appeal to lemmas and the restriction to reasoning only about closed expressions. We are presently extending both the meta-logic $\mathcal{M}_2$ and its implementation to overcome these limitations.

Example Suites. Twelf has been employed for a number of experiments in the area of programming languages and logics [Pfe96]. Many of these are contained in the example suite which is distributed with the Twelf system. Some of the examples contain fully automated proofs, others only their implementations as higher-level judgments.

One of the most well-developed case studies is Mini-ML. We prove value soundness, type preservation, and compiler correctness with respect to different abstract machines. This case study is fully explained and developed in [Pfe], as is a related development of pure logic programming. Other examples include natural deduction, axiomatic logical systems, sequent calculi for classical and intuitionistic logic, proofs of cut-elimination, Cartesian closed categories, and a proof of the Church-Rosser theorem.

2 Implementation

The implementation of Twelf comprises three major parts which we sketch in this section. At the heart is the *core type theory* which provides the necessary infrastructure for the representation of specifications, algorithms, and meta-theory. Algorithms can be executed by the *constraint logic programming* engine, and the development of the meta-theory is supported by the *meta-theorem proving* component.

Core Type Theory. The core of the implementation consists of a dependently typed λ-calculus λ^{Π} extended with notational definitions and existential variables. In contrast to earlier implementations, terms are represented in spine notation [PS98] with explicit substitutions, where we take advantage of normal forms in the $\lambda\sigma$-calculus. Composition of substitutions is a defined function rather than a constructor subject to rewrite rules. Presently, we have not yet undertaken the empirical analysis necessary for optimization, but we found explicit substitutions to be a useful organizing force in the structure of the implementation.

The most frequently used operations are weak head reduction and unification; they are employed for type reconstruction, logic program execution, and theorem proving. The simply-typed case of our unification algorithm is described in [DHKP96]; the dependently typed case is very similar. In short, equations which fall within the fragment of higher-order patterns are solved eagerly, while all other equations are postponed as constraints. Such constraints are indexed on their head variable and are reawakened when this variable is instantiated, which means no overhead for dormant constraints. Instantiation of variables during unification is implemented as an effect which is trailed so it can be undone during backtracking.

Twelf supports notational, non-recursive definitions. They are typically used as syntactic sugar for expressions or to abbreviate derivations of lemmas and theorems. Definitions are checked for strictness [PS98], which means we can often avoid expanding them during unification without sacrificing soundness or completeness. Briefly, a definition is strict if its expansion cannot discard any of its arguments. This guarantees that during unification definitions must only be expanded in the case that the heads of two expressions clash.

A separately implemented type checker is designed to verify the validity of terms in the core type theory without relying on complex algorithms for unification or search. It has been internally employed during the development of the Twelf system, and can be activated upon request when using Twelf.

Constraint Logic Programming. The logic programming interpreter for LF signatures works with a rudimentary compiled form. A compiled signature may be executed on an abstract machine based on a continuation-passing interpreter which maintains higher-order equational constraints. The implementation of this interpreter is presently straightforward and less efficient then the previous Elf implementation.

Twelf provides two tools to verify that the operational reading of a signature as a logic program is consistent: a mode checker and a termination checker. Both are helpful in the implementation of algorithms, since they allow static detection of common programming errors which escape the type-checker, such as misspelled variable names or incorrect subgoal ordering.

The mode checker verifies that the roles of input and output arguments to a predicate are respected throughout the program. The termination checker is given an extension of the subterm ordering on higher-order terms [RP96] and verifies that each well-moded query eventually either fails or yields a solution. Extensions can either be *lexicographic* or *simultaneous*. The implementation also allows termination orderings across several mutually recursive predicates.

A third tool to check that all cases for a set of input variables to a predicate are covered is currently in preparation. Together with mode and termination checking this can verify that a predicate implements a possibly non-deterministic function.

Meta-Theorem Proving. The meta-theorem proving module implements a special-purpose inductive theorem prover for deductive systems [SP98]. It requires the meta-theorem and a termination ordering which expresses the induction ordering, but is otherwise completely automatic. In particular, it does not support tactic-style or interactive theorem proving.

The prover chains together three basic operations: *filling*, *splitting*, and *recursion*. Filling uses an iterative-deepening variant of the constraint logic programming interpreter to perform direct search. Splitting performs complete case analysis based on the (possibly dependent) type of a variable. It requires unification and exploits a static subordination relation on type families to remove spurious parameter dependencies. Finally, there is recursion which appeals to the available induction hypotheses according to the given induction ordering.

The theorem prover generates explicit representations of the proofs it finds as higher-order judgments in LF. These are guaranteed to satisfy mode, termination, and coverage properties and can be safely executed as logic programs.

3 Environment

While Twelf is implemented in ML it is executed as a stand-alone program rather than within the ML top-level loop. This is feasible, since meta-programming is carried out in type theory itself via a logic programming interpretation, rather than in ML as in many other proof development environments. The most effective way to interact with Twelf is as an inferior process to Emacs. The Emacs interface, which has been tested under XEmacs, FSF Emacs, and NT Emacs, provides an editing mode for Twelf source files and commands for incremental type checking, logic program execution, and theorem proving. Moreover it provides utilities for jumping to error locations and tagging and maintaining configurations of source files.

The most sophisticated feature of the front end is type reconstruction, which is based on a duality between implicit quantification and implicit arguments, thereby reducing a type reconstruction problem to unification. With the unification algorithm sketched above, this means our system will always either report a principal type, a type error, or, in the case of unresolved constraints, an indication that the source term contained insufficient type annotations. This simple technique has shown itself to be surprisingly effective and often leads to precise error location information.

Acknowledgements. We would like to thank Iliano Cervesato for his contributions to the logic programming component of Twelf.

References

DHKP96. Gilles Dowek, Thérèse Hardin, Claude Kirchner, and Frank Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 259–273, Bonn, Germany, September 1996. MIT Press.

HHP93. Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

Pfe. Frank Pfenning. *Computation and Deduction.* Cambridge University Press. In preparation. Draft from April 1997 available electronically.

Pfe91. Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

Pfe94. Frank Pfenning. Elf: A meta-language for deductive systems. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, pages 811–815, Nancy, France, June 1994. Springer-Verlag LNAI 814. System abstract.

Pfe96. Frank Pfenning. The practice of logical frameworks. In Hélène Kirchner, editor, *Proceedings of the Colloquium on Trees in Algebra and Programming*, pages 119–134, Linköping, Sweden, April 1996. Springer-Verlag LNCS 1059. Invited talk.

PS98. Frank Pfenning and Carsten Schürmann. Algorithms for equality and unification in the presence of notational definitions. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *Types for Proofs and Programs.* Springer-Verlag LNCS, 1998. To appear.

RP96. Ekkehard Rohwedder and Frank Pfenning. Mode and termination checking for higher-order logic programs. In Hanne Riis Nielson, editor, *Proceedings of the European Symposium on Programming*, pages 296–310, Linköping, Sweden, April 1996. Springer-Verlag LNCS 1058.

SP98. Carsten Schürmann and Frank Pfenning. Automated theorem proving in a simple meta-logic for LF. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction (CADE-15)*, pages 286–300, Lindau, Germany, July 1998. Springer-Verlag LNCS 1421.

System Description: inka 5.0 - A Logic Voyager

Serge Autexier[1], Dieter Hutter[2], Heiko Mantel[2], and Axel Schairer[2]

[1] Saarland University, Fachbereich Informatik, Postfach 15 11 50
D-66041 Saarbrücken, Germany
autexier@cs.uni-sb.de

[2] German Research Center for Artificial Intelligence, Stuhlsatzenhausweg 3
D-66123 Saarbrücken, Germany
{hutter,mantel,schairer}@dfki.de

1 Introduction

Originally developed as an automatic inductive theorem prover [2] based on resolution and paramodulation, the inka system was redesigned in inka 4.0 in the early '90s [8] to meet the requirements arising from its designated use in formal methods. Meanwhile several large industrial applications of the verification support environment (VSE) [7] have been performed which gave rise to thousands of proof obligations to be tackled by its underlying deductive system inka.

The new version of inka is a result of this long experience made in formal software development. Thus, the major improvements of inka 5.0 are concerned with the requirements arising when dealing with large applications. The user database is distributed along different deductive units each of which consists of an individual logic (consequence relation) and a set of (local) axioms. In order to allow for the logical implementation of structured specifications as they are provided in languages like CASL [3], the deductive units may import the deductive reasoning of other units with the help of morphisms. Relationships between different units are also postulated with the help of morphisms between two units which give rise to various proof obligations. inka also supports the evolutionary aspect of formal software development as it incorporates a management of change. It minimizes the proof obligations arising when changing a deductive unit or defined relationships between some units. As a basis for the implementation of different logics, inka provides an annotated λ-calculus as an underlying meta-language. Annotations are a generalization of the colour concept [6] and are used to incorporate domain knowledge into the proof search process. inka provides a uniform hierarchical proof datastructure and a generic tactic definition mechanism to implement appropriate proof search engines.

2 System Description

Meta-level Language. inka supports the use of formulas of different logics and proof-objects of different calculi. All these objects are encoded in a meta-level language, based on a (ML-type) polymorphic λ-calculus [4]. λ-terms are automatically kept in $\beta\eta$ long normal-form. Using λ-calculus as meta-language provides a clear concept of variables, namely bound and free variables.

The annotations in inka's λ-calculus are described in a first-order term language and are attached to occurrences of function constants and variables, to λ-abstractions and applications. The concept of annotations is a generalization of the concept of colours [6], which has been developed in the context of inductive theorem proving to encode the knowledge about the similarities between induction hypothesis and induction conclusion [9]. They are used by tactics to encode domain knowledge into logical objects and the underlying annotated calculus supports the inheritance of the annotations. For a detailed description of annotated λ-calculus see [10,6].

The implementation of the annotated λ-calculus also comprises a unification for annotated λ-terms with free variables [10,6]. The unification algorithm is generic in the sense, that domain specific unification algorithms can be integrated into it. Thus, specialized unification algorithms, which make use of semantic properties of defined function constants, can be implemented and are integrated in an object-oriented manner into the generic unification algorithm.

Hierarchy of Logics. inka provides a mechanism for user defined logics in terms of so-called logic units (cf. Figure 1). The rôle of a logic unit is to define a truth-type and a language of logical formulas. In addition to the declarative content, a logic unit may contain specific domain-knowledge. For example, a logic unit can contain a specific theory unification algorithm for formulas; for first-order logic, a specific unification algorithm is implemented, which moves quantifiers over other connectives. Each time a logic unit is used, the specific unification is linked into the generic unification from the underlying λ-calculus. This approach differs essentially from logical frameworks like LF [5] or Isabelle [12] in which logics, respectively, are represented as signatures in a dependent type theory or by an embedding into a meta-logic.

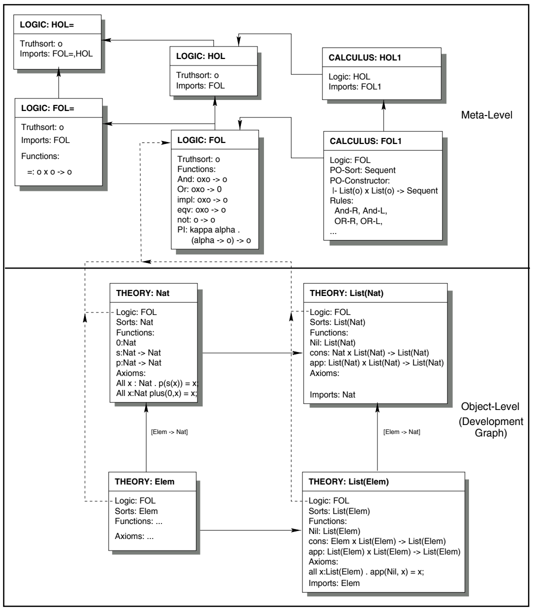


Fig. 1. The hierarchy of logics and the development graph

In inka, a calculus is represented within a calculus unit which is related to a logic unit and defines the proof objects as well as the basic calculus rules. E.g., in the sequent calculus unit for first-order logic, the proof-objects are sequents, which are build from lists of first-order logic formulas. The implementation of the basic calculus rules as well as the tactics for proof search are attached to the calculus units. If a calculus unit is used, its tactics are linked into the generic deduction mechanism which supports forward as well as backward application of rules. inka allows for an implementation of different calculi for different logics. The tactics and calculus rules are implemented using the generic tactic definition mechanism described in the paragraph Deduction. The proofs are constructed by a uniform tactic mechanism and are represented in an uniform proof datastructure (cf. Paragraph Deduction).

Furthermore, different kinds of relationships between logics are explicitly represented by logic morphisms. Along these morphisms, formulas of one logic can be transformed into equivalent formulas of another logic.

Development Graph. It has long been recognised that so-called *specifications in the large* are only manageable if they are built in a structured way on the basis of smaller specifications. Specification languages, like for instance CASL [3], provide various mechanisms to combine basic specifications to extensive structured specifications.

inka supports the use of such structured specifications and thus also the structured deduction by distributing the resulting logical axiomatization into so-called deductive units. Each deductive unit describes a logical theory and corresponds to a basic specification as defined in [3]. A deductive unit is linked to a calculus unit defining its underlying consequence relation. The units can be connected via consequence morphisms allowing the user to import (mapped) theories to a deductive unit. The same mechanism can be used to postulate relations between deductive units. Drawing a so-called theorem link from a unit N_1 to a unit N_2 (wrt. a morphism σ) gives rise to the proof obligation that the mapped theorems of N_1 are valid within N_2. inka provides several techniques to minimize the arising proof obligations by making use of already existing relations between other units linked to N_1 and N_2.

Deduction. The application of formal methods in an industrial setting (cf. [7]) results in an increased complexity of the specification and the correspondent verification. Tackling arising proof obligations from industrial case studies, the proofs are too complex to be done fully automatically but they are also too longish to be done by hand. Thus there is a need to combine a high degree of automation with an elaborate user interface to advise the deduction system in case where built-in strategies are too weak to find the proof automatically.

All deductions in the inka-system are explicitely represented in a uniform hierarchical proof datastructure, which is a generalization of the hierarchical proof datastructure (PDS) developed for the ΩMEGA-system [1]. It is realized by an acyclic directed graph, whose nodes are annotated by proof-objects defined by some calculus and whose edges are annotated by basic calculus rules or tactics. The lowest proof level consists of edges annotated only by basic calculus rules which are related to higher edges annotated by tactics. This datastructure allows to view a proof on different levels of abstractions. Hence, the proof can be communicated to a user on different levels of abstraction, which is an adequate mechanism to support interactive proof construction.

Tactics and calculus rules are implemented by a generic tactic definition mechanism, which allows for a simple implementation of tactics and rules and hides necessary updates of the proof graph from the tactic engineer. The tactic definition mechanism allows for the tactic engineer to focus on parts of a proof-object. Foci are explicitely represented and the designed tactics can manipulate the content of their argument foci, without changing the foci themself. This is a simple and efficient mechanism, offering the necessary level of abstraction to a tactic engineer to support tactic design. The tactic definition mechanism already existed in the old inka-system [8] and has been adapted to the hierarchical proof representation and in order to allow for the manipulation of foci wrt. different proof-objects within one tactic.

Interfaces. The user interface of the inka-system is an adaptation of the interface system $\mathcal{L}\Omega\mathcal{UI}$ [13], which is a user interface for the ΩMEGA-system [1]. $\mathcal{L}\Omega\mathcal{UI}$ has been adapted to be a generic user interface for theorem provers as for instance a visualization of the development graph has been added. It is implemented in the distributed Oz programming language [11] and the communication with inka is done via a socket-communication. The interface between Lisp and Oz is an abstract representation of the datastructures to be visualized while the actual layout is done in Oz. This minimizes the communication effort between Lisp and Oz and results in a sufficient speed of the visualization process.

The inka specification language is designed to include a subset of the *Common Algebraic Specification Language (CASL)* [3], which is currently developed to define a standard language for algebraic specifications.

3 Progress, Availability, and Future

The inka 5.0 system is an experiment in providing a generic software verification system. It is still in an incomplete, prototypical state. However, all basic mechanisms described in this paper are implemented and used for some example logics and sequent calculus proof search.

The core inka is implemented in Allegro Common Lisp. The interface runs on distributed Oz, which is available for Unix and Windows.

As a next step we intend to integrate a logic for algorithmic function and predicate definitions as well as the methods to prove their termination as tactics. Termination proofs can be inspected and already proven lemmata can be used during the construction of termination proofs, which are the main advantages wrt. the black box implementation of these methods in the old inka system [8].

References

1. C. Benzmüller et al.: ΩMEGA: Towards a mathematical assistant, In W. McCune (ed), *CADE-14*, Springer, LNAI 1249, 1997.

2. S. Biundo, B. Hummel, D. Hutter, C. Walther: The Karlsruhe Induction Theorem Proving System. In Jörg H. Siekmann (ed), *CADE-8*, Springer, LNCS 230, 1986.
3. CoFi-task group on language design, ESPRIT working group 29432, EU, 1998. CoFI Webpage: `http://www.brics.dk/Projects/CoFI/`
4. L. Damas, R. Milner: Principal type schemes for functional programs, *Ann. ACM Symp. on Principles of Programming Languages (POPL)*, 1982.
5. R. Harper, F. Honsell, G. Plotkin: A framework for defining logics, *Journal of the Association for Computing Machinery*, 40(1), 1993.
6. D. Hutter, M. Kohlhase: Managing Structural Information by Higher-Order Colored Unification, *Journal of Automated Reasoning*, accepted, 1999.
7. D. Hutter et al.: Verification Support Environment (VSE), *Journal of High Integrity Systems*, Vol. 1, 1996.
8. D. Hutter, C. Sengler: inka - The Next Generation. In M. McRobbie, J. Slaney (ed), *CADE-13*, Springer, LNAI 1104, 1996.
9. D. Hutter: Guiding Induction Proofs, In M. Stickel (ed), *CADE-10*, Springer, LNAI 449, 1991.
10. D. Hutter: Coloring terms to control equational reasoning, *Journal of Automated Reasoning*, Vol. 18, 1997.
11. Programming System Lab, Saarland University, Saarbrücken, 1998. The Oz Webpage: `http://www.ps.uni-sb.de/ns3/oz/`
12. L.C. Paulson: Isabelle, A Generic Theorem Prover, Springer, LNCS 828, 1994.
13. J. Siekmann et al.: A Distributed Graphical User Interface for the Interactive Proof System ΩMEGA. In R. C. Backhouse (ed), *UITP98*, Eindhoven Technical University, Report 98-08, 1998.

System Description: `CutRes 0.1`: Cut Elimination by Resolution*

Matthias Baaz[1], Alexander Leitsch[2], and Georg Moser[1]

[1] Institut f. Algebra und Computermathematik, E118.2, Technische Universität Wien
Wiedner Hauptstrasse 8–10, A-1040 Vienna, Austria
`{baaz,moser}@logic.at`

[2] Institut f. Computersprachen, E185.2, Technische Universität Wien
Resselgasse 3, A-1040 Vienna, Austria
`leitsch@logic.at`

Abstract. `CutRes` is a system which takes as input an **LK**-proof with arbitrary cuts and skolemized end-sequent and gives as output an **LK**-proof with atomic cuts only. The elimination of cuts is performed in the following way: An unsatisfiable set of clauses $\mathcal{C}$ is assigned to a given **LK**-proof Π. Any resolution refutation ψ of $\mathcal{C}$ then serves as a skeleton for an **LK**-proof Σ of the original end-sequent, containing only atomic cuts; Σ can be constructed from ψ and Π by projections. Note, that a proof with atomic cuts provides the same information as a cut-free proof.

1 Introduction

The use of lemmas in mathematical proofs lies at the very centre of mathematical reasoning; by the use of lemmas it is possible to structure the proof and hence to improve its readability and understandability. Therefore a real mathematical proof (modelled within a sequent calculus) always contains cuts; in the practice of mathematics the elimination of these cuts is, in most cases, not only useless but even destructive to intuition and mathematical information. However from a proof theoretic point of view cut-elimination plays a central rôle. A cut-elimination theorem for a sequent calculus has several important consequences (e.g. subformula property, interpolation theorem, etc.) and much effort traditionally is spent to prove such theorems in various calculi.

Our main motivation is automatization of mathematical reasoning. Viewing mathematical knowledge not as a knowledge of structures but as a knowledge of proof methods the enormous impact of the analysis

* This work has been sponsored by the Austrian Research Fund (FWF-grant No. P11934-MAT).

of proofs on the automatization and understanding of mathematics becomes obvious. Proof theory provided algorithms (encoding proof methods) since it very beginnings but they have only been applied in special cases [Gen34,Her71,Gir87], c.f. the analysis of van der Waerden's Theorem by Girard [Gir87]. In this paper we present a method of cut-elimination by resolution implemented as the system `CutRes`. We describe the current state of the system and intended extensions. The investigation of *cut formulas* and the impact of the *syntax of cuts* to cut-elimination and proof complexity have received relatively less attention. The system `CutRes` is meant as a tool to simplify such investigations.

A characteristic feature of the famous cut-elimination method by Gentzen [Gen34] is the stepwise reduction of cut-complexity. In this reduction the cut formulas are decomposed w.r.t. their out-most logical operator. Moreover the cut formulas to be eliminated must be rendered main formulas of inferences by adequate proof transformations. Despite its elegance the method of Gentzen is very costly, as it is largely independent of the *inner* structures of the cut formulas. These inner structures are the essence of cuts in real mathematics: mathematical arguments are typically based on *explicit* definitions, e.g. differentials, integrals etc. Therefore it is useful to concentrate on cut-elimination procedures which eliminate cuts by *analyzing* these explicit definitions and reducing cuts from inside out.

In [BL99] a new cut elimination method has been presented: This method characterises cuts by sets of clauses obtained from the derivation of the cut formulas. These sets of clauses are always unsatisfiable and thus have a resolution refutation. Any such refutation serves as skeleton of an **LK**-proof with only atomic cuts. `CutRes` is a system which takes as input a proof Π with skolemized end-sequent in Gentzen sequent calculus and produces a sequent calculus proof Σ with only atomic cuts. The output is in LaTeX-format. The limitation to skolemize the end-sequent is necessary for the correctness of the method, see [BL99]. Clearly it is easy to lift this restriction by skolemizing the proof as a pre-processing step and re-skolemize the final proof. See [BL94] for skolemization techniques not increasing the proof length. The main aim of `CutRes` —serving as a experimental tool for investigations on the effect of cut syntax on cut-elimination—is not affected by this restriction.

2 The System `CutRes`

The components of `CutRes` are written in Prolog[1]. The main task is to compute an unsatisfiable set of clauses $\mathcal{C}$ characterising the cut formulas. On this set of clauses a resolution theorem prover[2] is invoked. The resolution refutation ψ is re-translated to sequent notation and a ground-projection is computed. For each clause $C \in \mathcal{C}$ the system generates a projection Π_C—deriving the respective clause with additional side-formulas—out of the original proof Π. In the last step, ψ is extended with the projections $\Pi_C\colon C \in \mathcal{C}$ to obtain the final proof Σ.

The unsatisfiable set of clauses $\mathcal{C}$ for the example proof in table 1 consists of the following three clauses.

$$\to P(a) \quad \to Q(a) \quad P(X), Q(a) \to$$

The ground-projection of the resolution proof can be seen in the left-hand side of table 2. Table 3 shows the proof projection for the clause $P(X), Q(Y) \to$.

The output consists of three files containing LaTeX-definitions specific to (i) resolution proof, (ii) projections, and (iii) the final proof Σ. These files are included in a generic document when compiled with TeX. The files are generated by `CutRes` using DCGs (definite clause grammars) on the basis of a user-definable configuration file, which allows her to fine-tune the appearance of the proofs in the final document. The right-hand side of table 2 shows the overall structure of the generated document. The sections typed in *italic typeface* are sections whose content is generated by the system according to the input proof. With respect to the example proof Π the section *Proof Projections* will contain a proof projection for each clause above in the form shown in table 3.

3 Intended Extensions

In the present state the system eliminates all cuts simultaneously. However, if we wish to mimic Gentzen's cut elimination procedure it would be desirable to see the intermediate proof after eliminating the top-most cut. In theory this poses no problem, because each cut can be eliminated subsequently starting with the fragment of Π leading to the top-most cut. The result is then re-skolemized and combined with the original

[1] The system has been tested with SICStus Prolog, but should be portable to any standard Prolog

[2] The present version uses SPASS V0.86 as theorem prover, but any refutational theorem prover can be used.

Table 1. Table 1: Input proof Π with skolemized end-sequent

Π_0:

$$\dfrac{\dfrac{\dfrac{P(a)\to P(a)}{P(a)\to(\exists[x]\colon P(x))}}{(P(a)\land Q(a))\to(\exists[x]\colon P(x))} \quad \dfrac{Q(a)\to Q(a)}{(P(a)\land Q(a))\to Q(a)}}{(P(a)\land Q(a))\to((\exists[x]\colon P(x))\land Q(a))}$$

$$\dfrac{\overset{\Pi_0}{(P(a)\land Q(a))\to((\exists[x]\colon P(x))\land Q(a))} \quad \dfrac{\dfrac{\dfrac{\dfrac{P(\alpha)\to P(\alpha)\quad Q(a)\to Q(a)}{P(\alpha),Q(a)\to(P(\alpha)\land Q(a))}}{P(\alpha),Q(a)\to(\exists[x]\colon(P(x)\land Q(a)))}}{(\exists[x]\colon P(x)),Q(a)\to(\exists[x]\colon(P(x)\land Q(a)))}}{((\exists[x]\colon P(x))\land Q(a))\to(\exists[x]\colon(P(x)\land Q(a)))}}{(P(a)\land Q(a))\to(\exists[x]\colon(P(x)\land Q(a)))}$$

Table 2. Table 2: Resolution Proof and Output

$$\dfrac{\to P(a)\quad \dfrac{\to Q(a)\quad Q(a),P(a)\to}{P(a)\to}}{\to}$$

Cut Elimination by Resolution

C. Utres

1 *Original Proof*
2 *Resolution Proof*
3 *Proof Projections*
4 *Results*
References

Table 3. Table 3: Proof projection

The following is a projection out of the original **LK**-proof with respect to the clause $P(X), Q(a)\to$:

$$\dfrac{\dfrac{P(\alpha)\to P(\alpha)\quad Q(a)\to Q(a)}{P(\alpha),Q(a)\to(P(\alpha)\land Q(a))}}{P(\alpha),Q(a)\to(\exists[x]\colon(P(x)\land Q(x)))}$$

proof. It is obvious that such a method is less tractable. One tempting solution to this problem would be the replacement of Skolem terms by Hilbert's ϵ-terms, see [HB70]. In the presence of ϵ-terms no problem with the re-translation of the proof would occur. But it immediately turns out that ϵ-terms are much stronger then Skolem terms and cannot simply be disguised as Skolem terms under these circumstances. Another solution would be to generate the final proof Σ carefully, so to avoid an extraordinary increase in proof length. The upcoming goals for future versions of `CutRes` are mainly

◇ *Augmenting the cut elimination techniques*:
At present all cuts in the proof are eliminated simultaneously, according to the method in [BL99]. We will integrate a second approach where every cut is eliminated subsequently and the user has the possibility to view the intermediate steps.

◇ *Providing the user with more elaborate interfaces*:
At present the user has to write the **LK**-proof in an ordinary text file, which is then interpreted by the Prolog interpreter. We will provide her with a window-based environment that allows easier specification of the input. On the other hand we will provide an interface to WWW: The user fills in some HTML forms and gets the output of `CutRes` as a Postscript file, obviating the need to install on her own machine.

◇ *Providing the user with control over employed resolution refinements*:
At present the user has no control on the specific strategy used by the underlying theorem prover. However under specific circumstances a specific strategy and/or redundancy elimination techniques can prove useful. Because the strategy may incorporate additional information on the underlying mathematical theory.

References

BL94. M. Baaz and A. Leitsch. On Skolemization and proof complexity. *Fund. Inform.*, 20(4):353–379, 1994.

BL99. M. Baaz and A. Leitsch. Cut elimination by resolution. *J. Symbolic Computation*, 1999. To appear.

Gen34. G. Gentzen. Untersuchungen über das logische Schließen I–II. *Math. Zeitschrift*, 39:176–210, 405–431, 1934.

Gir87. J. Y. Girard. *Proof Theory and Logical Complexity*, volume 1 of *Studies in Proof Theory, Monographs.* Bibliopolis, Napoli, Italy, 1987.

HB70. D. Hilbert and P. Bernays. *Grundlagen der Mathematik 2.* Spinger Verlag, 1970.

Her71. J. Herbrand. *Jacques Herbrand: Logical Writings.* D. Reidel Publishing Company, Holland, 1971.

System Description: MathWeb, an Agent-Based Communication Layer for Distributed Automated Theorem Proving

Andreas Franke and Michael Kohlhase

FB Informatik, Universität des Saarlandes
{afranke,kohlhase}@ags.uni-sb.de

1 Introduction

Real-world applications of theorem proving require open and modern software environments that enable modularization, distribution, inter-operability, networking, and coordination. This system description presents the MathWeb[1] approach for distributed automated theorem proving that connects a wide-range of *mathematical services* by a common, *mathematical software bus.* The MathWeb system provides the functionality to turn existing theorem proving systems and tools into mathematical services that are homogeneously integrated into a networked proof development environment. The environment thus gains the services from these particular modules, but each module in turn gains from using the features of other, plugged-in components.

2 Implementation

The MathWeb system is an object-oriented toolbox that provides the functionality for building a society of software agents that render mathematical services by either encapsulating legacy deduction software or their own functionality. In the current implementation the software bus functionality is realized by a model quite similar to the *Common Object Request Broker Architecture* (CORBA [Sie96]) in which a central *broker* agent provides routing and authentication information to the mathematical services (see [SHS98] for details). The agents are realized in a distributed programming system mOZart[2], which provides the full infrastructure to write distributed applications.

Furthermore, MathWeb provides the mOZart shell (Mosh), a tool for launching and administering multiple mOZart applications (the agents) within only one mOZart process. It combines some frequently used shell commands (for files, processes and environment) with some (thread-related) mOZart commands. These allow (remotely) administering the mathematical services across the Internet, since the administrator can connect to remote Mosh demons[3],

[1] The system is available at `http://www.ags.uni-sb.de/~omega/www/mathweb.html`.
[2] See `http://www.mozart-oz.org`
[3] which run continually at the host providing the services

launch and terminate services. This also allows for a limited form of self-organization of mathematical services, since these can use MOSH scripts themselves to launch and administer other services.

mOZARTs main advantage as a basis for MATHWEB comes from its network transparency, i.e., the full support of remote computations in the base language (lexical scoping, logical variables, objects, constraints,...), and its network awareness, i.e., the full control over network operations, such as the choice between stationary and mobile objects, which make it easy to 'agentify' arbitrary applications.

3 Existing Mathematical Services

In this section we will briefly list and categorize the currently available mathematical services.

Automated Theorem Provers. MATHWEB currently features the first-order theorem provers BLIKSEM, EQP, OTTER, PROTEIN, SPASS, WALDMEISTER (see [SS98] for references), and the higher-order systems TPS [ABI+96] and $\mathcal{LEO}$ [BK98]. Furthermore, there is a service `competitive-atp` that calls sets of ATP concurrently as competing services (this strategy is known to yield even super-linear speedups in practice).

Computer Algebra Systems. There are services wrapping the systems MAPLE, MAGMA, GAP and μCAS (see [KKS98] for references). Here, the MATHWEB approach is particularly interesting, since a licensee of commercial software systems like MAPLE and MAGMA can export the corresponding services to the deduction community.

Mediators are mathematical services that transform mathematical knowledge from one format to another. The agent-oriented MATHWEB approach allows to encapsulate the zoo of conversion programs currently available[4] to generally available mathematical services and avoid duplication of efforts. Proof Transformers are rather substantial mediators that transform between proof formats. Currently MATHWEB features a proof transformation service from the proof formats of the theorem provers mentioned above [HF96,Mei99] to the natural deduction calculus.

Knowledge Bases. MATHWEB currently only includes the MBASE service, a simple web-based mathematical knowledge base system that stores mathematical facts like theorems, definitions and proofs and can perform type checking, definition expansion and semantic search. It communicates with other mathematical services by mediators and with humans by the interaction unit OCTOPUS.

Human Interaction Units are MATHWEB services that provide visualization and control features for the user interaction. Currently, MATHWEB includes the $\mathcal{L}\Omega\mathcal{UI}$ graphical user interface for interactive theorem provers [SHB+98], the OCTOPUS front-end for MBASE and the PROVERB proof presentation

[4] e.g. at the TPTP library `http://wwwjessen.informatik.tu-muenchen.de/~tptp`

system [HF96], which can transform ND proofs to natural language. The Doris system (see section 4) is a MathWeb client from outside the domain of deduction systems.

4 Applications and Experiences

Ωmega: The work reported in this paper originates in the development effort of the Ωmega-system [BCF+97], a mathematical assistant system with the ultimate goal of supporting theorem proving in main-stream mathematics and mathematics education. To provide the necessary reasoning and symbolic computation facilities this system incorporates most of the mathematical services listed in section 3. The MathWeb approach has been a key factor in keeping the system maintainable [SHS98,FHJ+99] and the near future will see further modularization and agentification of system components, which will lead to simpler system maintenance and a more open development model.

Doris: Apart from this application, MathWeb has been tested in the Doris[5] system, a natural language understanding system that uses first-order automated theorem provers and model builders as external mathematical services to solve the consistency and entailment problems pertaining to various disambiguation problems in text and dialogue understanding. Doris generates between 1 and ca. 500 deduction problems for each sentence it processes, distributes them to competing mathematical services (over a network of workstations) and collects the results to obtain the desired result. Using the MathWeb approach, the integration of the theorem provers was very simple: the only new parts were a socket connection from Prolog on the Doris side and a new service module for the `doris` service[6] on the MathWeb side. Experience with this application shows that distribution using MathWeb does not come for free: A test with ca. 1300 Doris deduction queries yielded the following timings:[7]

30–1250 ms pure theorem proving time
50-120 ms spent in the service module (opening an inferior shell, creating files,...). This depends strongly on the efficiency of the server file system.
5–500 ms Internet latency (we have measured inter-department (in Saarbrücken) and international (Saarbrücken/Amsterdam) connections)

However, the large number of deduction problems and the possibility of coarse-grained parallelization by distribution lead to a significant increase in overall system performance, compared to an earlier centralized, sequential architecture.

[5] See `http://www.coli.uni-sb.de/~bos/atp/doris.html` for a web-based interface that acts as a MathWeb client

[6] I.e. a small (60 line) mOZart program that relays problems, results and statistics between the Doris program and the `competitive-atp` service.

[7] These times have been measured on a collection of SUN Ultra machines running Solaris 5 in Saarbrücken and Amsterdam (all timings given in total elapsed time; normalized to our fastest machine, a SUN Ultra 4 at 300 MHz).

In particular, the timings also show that it can pay off for a client in Saarbrücken to delegate deduction problems to faster machines in Amsterdam or vice versa.

5 Conclusion and Future Work

The MathWeb system provides a transport layer for distributed theorem proving and a set of mathematical services, which will grow over time. The authors would like to encourage the automated deduction community to supply further mathematical services.[8]

The current CORBA-like distribution model in MathWeb is sufficient in an agent society, where services and their abilities are relatively fixed and well-known, which is reasonable for the relatively closed projects described in section 4. As the number of available services will grow (MathWeb has for instance been adopted by other projects building on Doris), this design will become too inflexible. Therefore the logical next step will be to adopt a more general truly agent-based approach. We have started to extend MathWeb so that it uses the Kqml interlingua (*Knowledge Query and Manipulation Language* [FF94]) as the agent interaction language and the OpenMath [Cap98] standard as a content language.

This move will result in a "plug-and-play" architecture for theorem proving and (in the future) for doing mathematics and program verification on the web.

References

ABI+96. Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

BCF+97. C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. *Ω*mega: Towards a mathematical assistant. In William McCune, editor, *Proceedings of the 14th Conference on Automated Deduction*, number 1249 in LNAI, pages 252–255, Townsville, Australia, 1997. Springer Verlag.

BK98. Christoph Benzmüller and Michael Kohlhase. LEO – a higher order theorem prover. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the 15th Conference on Automated Deduction*, number 1421 in LNAI, pages 139–144, Lindau, , Germany, 1998. Springer Verlag.

Cap98. The open math standard. Open Math Consortium, `http://www.openmath.org`, 1998.

FF94. T. Finin and R. Fritzson. KQML — a language and protocol for knowledge and information exchange. In *Proceedings of the 13th Intl. Distributed Artificial Intelligence Workshop*, pages 127–136, Seattle, WA, USA, 1994.

FHJ+99. Andreas Franke, Stephan M. Hess, Christoph G. Jung, Michael Kohlhase, and Volker Sorge. Agent-oriented integration of distributed mathematical services. *Journal of Universal Computer Science*, 1999. forthcoming.

[8] Either by using MathWeb directly, or by cooperating with the authors.

HF96. Xiaorong Huang and Armin Fiedler. Presenting machine-found proofs. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th Conference on Automated Deduction*, number 1104 in LNAI, pages 221–225, New Brunswick, NJ, USA, 1996. Springer Verlag.

KKS98. Manfred Kerber, Michael Kohlhase, and Volker Sorge. Integrating computer algebra into proof planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

Mei99. Andreas Meier. Translation of automatically generated proofs at assertion level. Technical Report forthcoming, Universität des Saarlandes, 1999.

SHB^{+}98. Jörg Siekmann, Stephan Hess, Christoph Benzmüller, Lassaad Cheikhrouhou, Detlef Fehrer, Armin Fiedler, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, and Volker Sorge. A distributed graphical user interface for the interactive proof system OMEGA. In Roland C. Backhouse, editor, *User Interfaces for Theorem Provers*, number 98-08 in Computing Science Reports, pages 130–138, Department of Mathematics and Computing Science, Eindhoven Technical University, 1998.

SHS98. M. Kohlhase, S. Hess, Ch. Jung and V. Sorge. An implementation of distributed mathematical services. In *6th CALCULEMUS and TYPES Workshop*, Eindhoven, The Netherlands, July 13–15 1998. Electronic Procedings: `http://www.win.tue.nl/math/dw/pp/calc/proceedings.html`.

Sie96. Jon Siegel. *CORBA: Fundamentals and Programming*. John Wiley & Sons, Inc., 1996.

SS98. Geoff Sutcliffe and Christian Suttner. The cade-14 atp system competition. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

System Description
Using OBDD's for the Validation of Skolem Verification Conditions

E. Pascal Gribomont and Nachaat Salloum

Institut Montefiore, ULg, Sart-Tilman, B 28, B – 4000 Liège (Belgium)
Phone: [+]32 4 366 26 67, Fax: [+]32 4 366 29 84
gribomont@montefiore.ulg.ac.be

Abstract. The verification conditions associated with concurrent systems and their invariants are usually instances of the *VC-scheme*, i.e.,

$$(\bigwedge_{i=1}^{n} h_i) \Rightarrow c\,.$$

Besides, the following assumptions are often satisfied :
1. The set of hypotheses $H = \{h_1, \ldots, h_n\}$ is rather large;
2. The hypotheses and the conclusion are small quantifier-free formulas;
3. They are based on a large set of booleans and a smaller set of predicates;
4. If $H \models c$, then there is a small subset $H_0 \subset H$ such that $H_0 \models c$.
We demonstrate a specific, OBDD-based technique for validating instances of the VC-scheme. The main task of the tool is to construct an upper bound for H_0, as tight as possible. The technique is illustrated with an example.

1 Skolem Verification Conditions

A *concurrent system* $\mathcal{S}$ consists of a set of variables, a set of processes and a set of transitions. A transition specifies a computation step. An *assertion* is a relation between the variables of the program. If A is an assertion and if τ is a transition of the program, then $wlp[\tau; A]$ is the associated weakest precondition; if it is satisfied just before transition τ is executed, then assertion A is true just afterwards. For instance, $wlp[B[i] := 0;\ B[5] = 0]$ is $(i = 5 \vee B[5] = 0)$. An invariant is an assertion I which is respected by every transition : I is an invariant if and only if the implication $I \Rightarrow wlp[\tau; I]$ is valid for each τ. (See e.g. [4] for more details.) All safety properties of a program are logical consequences of an appropriate invariant, so invariant validation is the biggest part of program verification. An invariant is usually structured as a conjunctive set of assertions. Most of them are small, elementary formulas. Let $I = \bigwedge_{i=1}^{n} a_i$ be an invariant for a concurrent system $\mathcal{S}$. If a'_k denotes $wlp[\tau; a_k]$, then the formula $(\bigwedge_{i=1}^{n} a_i) \Rightarrow a'_k$

is a *verification condition* associated with τ and I. It is clearly an instance of VC-scheme.[1] So, software verification would benefit from fast analysers for instances of this scheme. Such instances also appear in hardware verification, in deductive databases, expert systems, logic programs, and so on.

The size of an appropriate invariant for a concurrent system tends to be as large as the system code. Each assertion of the invariant involves only a few variables and place predicates. Most variables used by concurrent systems (e.g., place predicates, flags and semaphores) are boolean. As a result, the atoms of the verification conditions are mostly boolean. Few predicates occur, usually without quantification. A frequent exception is the case of an arbitrary number of symmetric (or nearly symmetric) processes; the assertions about them usually are universally quantified formulas where the range of the quantification is the set of processes. All these particulars lead to the four assumptions listed in the Abstract. The fourth assumption is especially important: if we can guess which hypotheses are really needed to establish the conclusion, the nondeterministic search for a proof will become drastically easier. The main purpose of our tool is to automate the discovery of the k useful hypotheses in conditions like $(\bigwedge_{i=1}^{n} h_i) \Rightarrow c$, that is, to solve the *hypothesis selection problem* in the case of Skolem verification conditions (SVC).

2 An OBDD-Based Deductive Selection Method

2.1 The Method

The formulas investigated here are rather similar to propositional formulas, except that they contain some atoms written in a first-order language, usually arithmetic. Several methods and tools developed for pure propositional logic are likely to be adaptable for SVC validation. Here the OBDD[2] form for formulas is used to solve the hypothesis selection problem for SVCs.

The principle of the method is elementary. Given the instance $(\bigwedge_{i=1}^{n} h_i) \Rightarrow c$, the OBDD associated with the conclusion c is constructed first. Afterwards, the n OBDD's associated with the implications $(h_1 \Rightarrow c), \ldots, (h_n \Rightarrow c)$ are considered, and a minimal one, say $c_1 =_{def} (h_{j_1} \Rightarrow c)$, is selected.[3] The next phase is to construct the $n-1$ OBDD's associated with the $(h_i \Rightarrow (h_{j_1} \Rightarrow c))$, for all $i \neq j_1$, and to select a minimal one, say $c_2 =_{def} (h_{j_2} \Rightarrow (h_{j_1} \Rightarrow c))$. Similar phases take place until the one-node BDD corresponding to *true* is obtained. (In the sequel, *true* denotes both the trivially valid formula and the associated, one-node BDD.) If this occurs for the formula $c_k =_{def} (h_{j_k} \Rightarrow (\cdots \Rightarrow (h_{j_1} \Rightarrow c) \cdots))$, we know for sure that

[1] Recall that wlp is $\wedge$-additive: $wlp[\tau; \bigwedge_{i=1}^{n} a_i]$ reduces to $\bigwedge_{i=1}^{n} wlp[\tau; a_i]$, and the big verification condition $(\bigwedge_{i=1}^{n} a_i) \Rightarrow wlp[\tau; \bigwedge_{i=1}^{n} a_i]$ can be split into the more elementary conditions $(\bigwedge_{i=1}^{n} a_i) \Rightarrow wlp[\tau; a_j]$, $j = 1, \ldots, n$.

[2] We assume the reader has a working knowledge of (O)BDD's, that can be found in any introductory paper or tutorial, e.g. [1,3]

[3] An OBDD $x \in S$ is *minimal* if for all $y \in S$, $|x| \leq |y|$; $|x|$ denotes the *size* of x, i.e., the number of nodes in x.

– The formula c_k is valid;
– The initial formula is also valid, so $\{h_1, \ldots, h_n\} \models c$;
– Members of $\{h_1, \ldots, h_n\} \setminus \{h_{j_1}, \ldots, h_{j_k}\}$ are useless hypotheses;
– Hypothesis h_{j_k} is useful.

We hope that most of the hypotheses $h_{j_1}, \ldots, h_{j_{k-1}}$ are useful too. The idea behind this algorithm is simply that a useful hypothesis h decreases the "distance" between the conclusion c and the formula *true*. As the size of *true* is 1, we view progress towards validity as decreasing of the OBDD size. The sequence $c = c_0, c_1, \ldots, c_k = \mathit{true}$ is monotonic: if $i < j$ then $\models (c_i \Rightarrow c_j)$.

2.2 The Predicate Case

The OBDD approach extends rather easily to the quantifier-free predicate case. Let us consider the example of a formula Ψ whose all atoms are boolean, except $x < y$ and $x \geq y$. The verification system views these arithmetical atoms as distinct, semantically unrelated propositions, so even if Ψ is valid, the corresponding OBDD will probably not reduce to the trivial, valid OBDD. However, if the additional hypothesis $h =_{def} \neg(x < y \equiv x \geq y)$ is taken into account, validity will be detected; the OBDD associated with $(h \Rightarrow \Psi)$ will be *true*.

In practice, we apply the same selection algorithm in the propositional case and in the predicate case, except that, when available, additional hypotheses like $\neg(x < y \equiv x \geq y)$ are added first. If all necessary hypotheses have been added, the predicate case reduces to the propositional case. Otherwise, after every phase, i.e., every time a new hypothesis has been selected, or at intervals scheduled by the user, the verification system enters in interactive mode and repeatedly displays truthvalue assignments for the predicate atoms, which may falsify the formula c. The user has to decide if such a propositional assignment is possible. If yes, then the computation goes on, further hypotheses are selected and further troublesome assignments are displayed. If no, the user issues a valid additional hypothesis (connection), say h, which makes the displayed assignment impossible. This hypothesis is used to simplify the current OBDD, that is, c_j is replaced by $c_{j+1} =_{def} (h \Rightarrow c_j)$.

3 Example: Szymanski's Algorithm

Szymanski's algorithm for mutual exclusion between `n` concurrent processes [7,6] has often been used as a benchmark for verification systems. The description of the algorithm is not repeated here, neither is the long list of verification conditions. Instead, we will explain our method with the transition

```
p3[i] if (FA j : not s[j]) then begin skip end p4[i]
```

`s[j]` is a boolean variable owned by process `j`. This transition allows process `i` to switch from location `p3` to location `p4` when all `s[j]`'s are false.

The first step of our tool is the elimination of quantifiers from both the invariant (not listed here) and the transitions. For instance, assume there is in some transition a quantified guard like `(FA j : s[j])`, where each `s[j]` is a boolean variable owned by process `j`. A counter `#s` is introduced, which records the number of true `s[j]`'s, and the guard becomes `(#s = n)`. All assignments that modify some `s[j]` are rewritten in such a way they also update counter `#s`.

The second step is the construction of the verification conditions; it is straightforwards since all quantifications have been eliminated. For our version of Szymanski's algorithm, one of the verification condition is

```
        (H01 and ... and H44) ==> C
```

with

```
H44 (at_p8[j] ==> (at_p10[k] or at_p10[l]))
H43 (at_p8[k] ==> (at_p10[j] or at_p10[l]))
H42 (at_p8[l] ==> (at_p10[j] or at_p10[k]))
H41 ((at_p9[j] or at_p10[j] or at_p11[j] or at_p12[j]) ==> (not at_p4[k]))
H40 ((at_p9[j] or at_p10[j] or at_p11[j] or at_p12[j]) ==> (not at_p4[l]))
H39 ((at_p9[k] or at_p10[k] or at_p11[k] or at_p12[k]) ==> (not at_p4[j]))
H38 ((at_p9[k] or at_p10[k] or at_p11[k] or at_p12[k]) ==> (not at_p4[l]))
H37 ((at_p9[l] or at_p10[l] or at_p11[l] or at_p12[l]) ==> (not at_p4[j]))
H36 ((at_p9[l] or at_p10[l] or at_p11[l] or at_p12[l]) ==> (not at_p4[k]))
H35 ((at_p11[j] or at_p12[j]) ==> #nw=n)
H34 ((at_p11[k] or at_p12[k]) ==> #nw=n)
H33 ((at_p11[l] or at_p12[l]) ==> #nw=n)
H32 (((at_p12[j] or (at_p11[j] and (#ns mod (2**(j-1))=(2**(j-1))-1))) and (k<j))
     ==> (at_p0[k] or at_p3[k]))
H31 (((at_p12[k] or (at_p11[k] and (#ns mod (2**(k-1))=(2**(k-1))-1))) and (j<k))
     ==> (at_p0[j] or at_p3[j]))
H30 (((at_p12[k] or (at_p11[k] and (#ns mod (2**(k-1))=(2**(k-1))-1))) and (l<k))
     ==> (at_p0[l] or at_p3[l]))
H29 (not w[i])
H28 ((at_p5[j] or at_p6[j] or at_p7[j] or at_p8[j] or at_p9[j]) ==> w[j])
H27 (w[j] ==> (at_p5[j] or at_p6[j] or at_p7[j] or at_p8[j] or at_p9[j]))
H26 ((at_p5[k] or at_p6[k] or at_p7[k] or at_p8[k] or at_p9[k]) ==> w[k])
H25 (w[k] ==> (at_p5[k] or at_p6[k] or at_p7[k] or at_p8[k] or at_p9[k]))
H24 ((at_p5[l] or at_p6[l] or at_p7[l] or at_p8[l] or at_p9[l]) ==> w[l])
H23 (w[l] ==> (at_p5[l] or at_p6[l] or at_p7[l] or at_p8[l] or at_p9[l]))
H22 (not s[i])
H21 ((at_p5[j] or at_p6[j] or at_p9[j] or at_p10[j] or at_p11[j] or at_p12[j])==> s[j])
H20 (s[j]==> (at_p5[j] or at_p6[j] or at_p9[j] or at_p10[j] or at_p11[j] or at_p12[j]))
H19 ((at_p5[k] or at_p6[k] or at_p9[k] or at_p10[k] or at_p11[k] or at_p12[k])==> s[k])
H18 (s[k]==> (at_p5[k] or at_p6[k] or at_p9[k] or at_p10[k] or at_p11[k] or at_p12[k]))
H17 ((at_p5[l] or at_p6[l] or at_p9[l] or at_p10[l] or at_p11[l] or at_p12[l])==> s[l])
H16 (s[l]==> (at_p5[l] or at_p6[l] or at_p9[l] or at_p10[l] or at_p11[l] or at_p12[l]))
H15 (a[i])
H14 ((not at_p0[j]) ==> a[j])
H13 (a[j] ==> (not at_p0[j]))
H12 ((not at_p0[k]) ==> a[k])
H11 (a[k] ==> (not at_p0[k]))
H10 ((not at_p0[l]) ==> a[l])
H09 (a[l] ==> (not at_p0[l]))
H08 (#anw>0)
H07 ((at_p3[j] or at_p4[j] or at_p10[j] or at_p11[j] or at_p12[j]) ==> (#anw>0))
H06 ((at_p3[k] or at_p4[k] or at_p10[k] or at_p11[k] or at_p12[k]) ==> (#anw>0))
H05 ((at_p3[l] or at_p4[l] or at_p10[l] or at_p11[l] or at_p12[l]) ==> (#anw>0))
H04 ((at_p10[j] or at_p11[j] or at_p12[j]) ==> (#snw>0))
H03 ((at_p10[k] or at_p11[k] or at_p12[k]) ==> (#snw>0))
H02 ((at_p10[l] or at_p11[l] or at_p12[l]) ==> (#snw>0))
H01 ((#snw>0) ==> (at_p10[j] or at_p11[j] or at_p12[j] or at_p10[k] or
                   at_p11[k] or at_p12[k] or at_p10[l] or at_p11[l] or at_p12[l]))

C   ((#ns=n) ==> (not (at_p9[j] or at_p10[j] or at_p11[j] or at_p12[j])))
```

Only hypothesis `H21` is needed to establish the conclusion `C`. The system selects it immediately, provided the additional connection `(#ns=n) ==> not s[j]` is available. If not, the system cannot validate the verification condition but it orders the hypotheses; the useful one is selected early (in the fourth position), so the discovery of the adequate connection is fairly easy.

4 Global Results

The verification set associated with our version of Szymanski's algorithm and of the adequate invariant contains 70 non trivial verification conditions. Each of them is an instance of VC-scheme, with an average number of 42 hypotheses.

We considered the favourable case first and assumed all additional hypotheses (connections) about the arithmetical atoms of the program and the invariant had been included first. For 59 out of the 70 conditions, only the first selected hypothesis was needed to validate the conclusion. For the other conditions, we observed that the number of selected hypotheses did not exceed much the number of useful hypotheses; this indicates that the selection policy is appropriate.

We assumed afterwards the most unfavourable, and somewhat unrealistic case, where no additional hypothesis were known *a priori*; we also supposed that the user was not able to provide them interactively. As a consequence, conditions which are only "pseudo-tautologies" were not validated. Nevertheless, the verification system ordered the hypotheses. For 53 conditions, the only useful hypothesis was selected first. For 13 conditions, the useful hypotheses were among the first ten selected ones. Only for the 4 remaining conditions the selection policy was not very good and one of the useful hypotheses was selected late.

These results clearly show that the selection policy is a useful addition to our verification tool CAVEAT [5], but even a few pathologic cases are sufficient to prevent an easy and full verification. There is no surprise here : if an algorithm relies on some arithmetical facts, the verifier should know them.

References

1. H.R. Andersen, An Introduction to Binary Decision Diagrams, Available at `http://andrea.it.dtu.dk/~hra/notes-index.html` (1997, 1998).
2. A. Borälv, The Industrial Success of Verification Tools Based on Stalmårck's method, *Lect. Notes in Comput. Sci.* **1254** (1997) 7-10.
3. R.E. Bryant, Graph-based algorithms for Boolean function manipulation, *IEEE Trans. on Computers* **C-35** (1986) 677-691.
4. K.M. Chandy and J. Misra, *Parallel Program Design : A Foundation* (Addison-Wesley, Reading, MA, 1988).
5. E.P. Gribomont and D. Rossetto, CAVEAT : technique and tool for Computer Aided VErification And Transformation, *Lect. Notes in Comput. Sci.* **939** (1995) 70-83.
6. E.P. Gribomont and G. Zenner, Automated verification of Szymanski's algorithm, *Lect. Notes in Comput. Sci.* **1384** (1998) 424-438.
7. B. Szymanski and J. Vidal, Automatic Verification of a Class of Symmetric Parallel Programs, *13th IFIP World Computer Congress*, IFIP-Elsevier (1994) 571-576.

Fault-Tolerant Distributed Theorem Proving

Jason Hickey*

Cornell University

1 Introduction

In recent years, there have been many examples of significant formalization efforts in higher-order logics, including the formalization of Java [12], the verification of the SCI cache-coherency protocol [5], the verification of the AAMP5 avionics processor [11] in PVS [2], the verification and automated optimization of Ensemble protocols [10,7], and many others. Higher-order logics are often chosen for these endeavors not only because they can formalize meta-principles, but also because they retain the conciseness and intuition of the original design.

While this model has been successful, it is expensive. We present an architecture, implemented in the MetaPRL logical framework, for distributing tactic proving over large groups of processors using the Ensemble group communication system. To counteract problems of reliability in distributed environments, we use Ensemble's fault recovery support. To preserve the large existing tactic base, we replace the existing tactic implementation with a functionally equivalent distributed tactic scheduler.

2 Related Work

MetaPRL [8] is derived from the Nuprl proof development system, which includes both a logic and a mechanism for reasoning. An early version of the system is described in Constable et. al. [4]; more recent descriptions can be found in Jackson's thesis [9]. An account of Ensemble can be found in Hayden [6]. Ensemble and MetaPRL are both implemented in OCaml [13].

There are several examples of first-order proving and limited higher-order distributed theorem provers [1,3]. While these systems have some similarity to our work, we focus on general-purpose tactic proving, which differs in several major respects. First, the logic is not fixed, and our distribution mechanism must work for *any* definable logic. Second, tactic provers are designed for use in logics where proofs may need interactive guidance. This requires two features: the distributed prover must be tolerant to process failures, and proof search should be *deterministic*. Finally, tactics are higher-order search algorithms, and their distribution requires that arbitrary functions be passed along communication channels.

* Support for this research was provided by DARPA grant F30602-95-1-0047.

3 Refinement Architecture and Tactics

The MetaPRL proving architecture is implemented in several parts. There is a logic engine, called the *refiner*, for applying a primitive rule to a goal to reduce it to a (possibly empty) set of subgoals, there is a *primitive tactic* module that defines the basic tactic operations, and additional proof automation is implemented in the *tactic library*.

The refiner exports the operations in the OCaml signature below. The `term` type is used to represent logical sentences. The `rule` and `logic` types are used to represent primitive rules and logics (more detail about the construction of logics can be found elsewhere [8]). The `proof` type represents *partial* proof trees that have been constructed by refinement; the `refine` function applies a `rule` to a goal to produce a partial proof. The `compose` function is used to compose partial proof trees. If a tactic application fails, the `RefineError` exception can be raised.

The tactic-primitives module defines proof search with *tacticals*. The tactic-primitives module re-implements the `proof` type to contain single node proof trees, and it implements two *tacticals:* the `(andthenT tac1 tac2)` tactic applies `tac1` to the goal, then applies `tac2` to each of the subgoals. The `(orelseT tac1 tac2)` tactic applies `tac1` to the goal. If it succeeds, the proof is returned; otherwise, `tac2` is applied instead.

```
module Refine : sig
  type term, rule, logic, proof
  exception RefineError
  val refine : logic → rule → term → proof
  val compose : proof → proof list → proof
  val goal_of_proof : proof → term
  val subgoals_of_proof : proof → term list
end
```

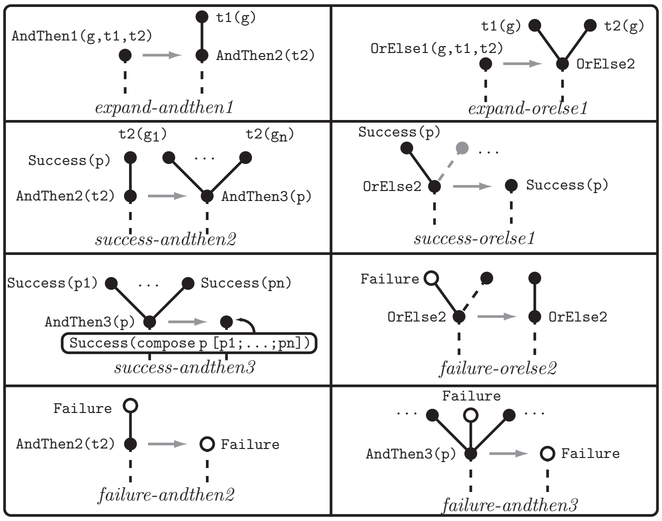
```
module TacticPrimitives (Refiner) : sig
  open Refiner
  type prim_tactic, proof
  type tactic = term → prim_tactic
  val tactic_of_rule : rule → tactic
  val proof_of_goal : logic → term → proof
  val refine : proof → tactic → proof
  val andthenT : tactic → tactic → tactic
  val orelseT : tactic → tactic → tactic
end
```

4 Distributed Refinement

```
AndThen1 (goal, tac1, tac2)
AndThen2 (tac2)
AndThen3 (p)
OrElse1 (goal, tac1, tac2)
OrElse2
Success (p)
Failure
```

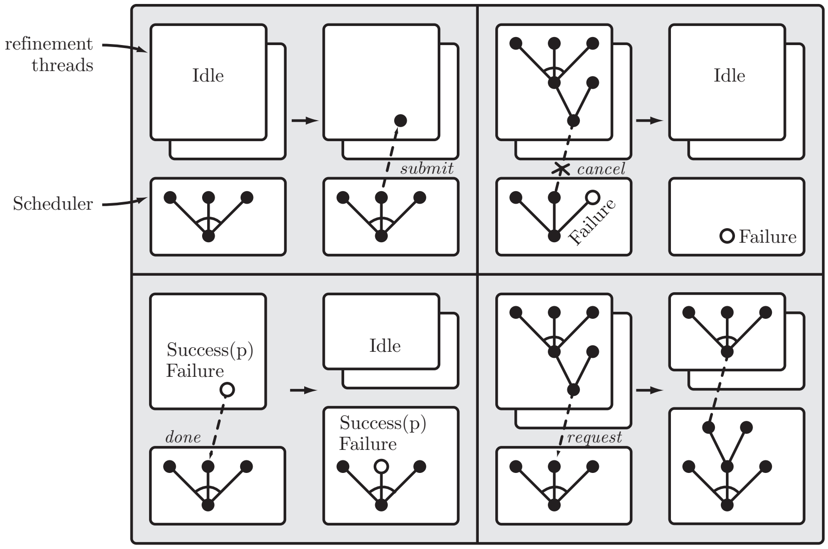
A tactic constructs an and-or branching proof tree, in which each node in the tree is labeled by a goal sentence and a tactic that was applied to the goal to produce the children. MetaPRL labels nodes in a proof tree with seven kinds of labels. The `AndThen` and `OrElse` labels are used for partially expanded nodes, and the `Success` and `Failure` labels are used when expansion is complete. Note that `Success` signals tactic completion, but the proof may still be partial.

The complete set of operations on proof trees is shown graphically in the first figure on the next page. The operations define expansion of `AndThen1` and `OrElse1` nodes, and they define how to back-propagate `Success` and `Failure` labels. A tree is complete when the root node is labeled either `Success(p)` for some proof `p`, or `Failure`. Note that two operations, *failure-andthen2* and *success-orelse1*, cause pruning of sibling nodes in the proof tree.



MetaPRL implements proof tree expansion in two parts. There are multiple *threads* that expand independent subtrees, and a *Scheduler* to issue tasks to threads. Each thread provides three functions to the scheduler: one to `submit` a node to be expanded, another to `request` the root node from the running thread, and another to `cancel` the subtree expansion. The result of an expansion is returned through the event produced by the `submit` function; the `cancel` function causes `Failure` to be returned immediately.

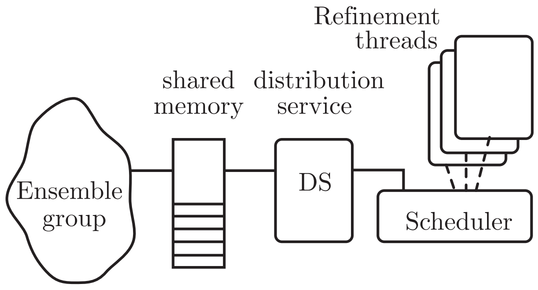
```
module type ThreadSig = sig
  type thread
  val submit : thread → node → node event
  val request : thread → unit
  val cancel : thread → unit
end
```



The operations implemented in the Scheduler are shown graphically in the figure above. The Scheduler submits a frontier node to an idle thread with the *submit* operation, and it receives the result from a completed thread with the *done* operation. When a branch of the tree is pruned, a thread may be terminated with the *cancel* operation. If there are idle threads and no available frontier nodes to be scheduled, the scheduler can request that a thread return the root part of its tree to the scheduler with the *request* operation.

In distributed refinement, tactic evaluation occurs on multiple processors, and communication between refinement processes is through message passing. Our implementation of distributed refinement is symmetric; for each refiner process, we add an additional thread, called a *distribution service* (DS), that requests frontier proof nodes from the Scheduler and passes them to remote refiners.

For communication, we implement a shared-memory model using the broadcast mechanism of the Ensemble group communication system. Entries may be *locked* or *unlocked.* Locks are exclusive; if an entry is locked, it is locked by exactly one process. Entries are created in the unlocked state, and the process that created the entry is called its *owner.* The owner may delete an entry, even if it is locked; the lock holder is notified of the cancelation.



The DS emulates a refinement thread interface. When the scheduler `submits` a node to the DS, the DS stores the node in the shared memory. If the Scheduler `cancels` the node, the DS deletes the entry from the shared memory. When the Scheduler makes a `request` for a new node, the DS attempts to lock an entry in the shared memory. If the lock is successful, the subgoal associated with the entry is passed to the Scheduler as a new frontier node. If the entry is deleted, the Scheduler `cancels` the node associated with the entry. When the Scheduler completes the evaluation, it passes the result back to the DS, which returns the result to the owner.

Each process keeps a local copy of the shared memory, and Ensemble is used to broadcast new entries, and ensure that the copies are consistent. If a process fails, all entries owned by that process are deleted.

5 Performance Measurements

We give five examples for performance. The `Pigeon3` and `Pigeon4` examples are problems in propositional logic, the `Gen` example is a proof of heredity in a large geneological database in first-order logic, and the `Term` and `CZF` examples are proof search problems in the Nuprl type theory. In the table, the "unthreaded" column gives performance numbers for a single-threaded version of MetaPRL. The "distributed" column gives times for distributed proving using one to five processes (on different processors). The "failure" column is the time to solve the problem using five processes, where one of the processes fails about halfway through the search. All times are in seconds, for Pentium II 200Mhz processors running Linux.

The speedup on these examples is fairly consistent, averaging about 3.2 with for the 5 process case. The GEN problem is an exception; it acheived a superlinear speedup. On this problem, the MetaPRL default random scheduling algorithm performs better than the default depth-first-search.

Problem	unthreaded	distributed					failure
		1	2	3	4	5	
`Pigeon3`	7	8.6	7.2	6.2	4.5	3.8	
`Pigeon4`	207	221	141	95	80	67	95
`Gen`	109	118	73	41	28	20	45
`Term`	422	440	250	201	160	137	180
`CZF`	63	75	40	32	26	22	47

References

1. Maria Paola Bonacina and William McCune. Distributed theorem proving by peers. In *1194 Conference on Automated Deduction (CADE12)*, pages 841–845, 1994.
2. Judy Crow, Sam Owre, John Rushby, Natarajan Shankar, and Mandayam Srivas. A Tutorial Introduction to PVS. In *WIFT '95: Workshop on Industrial-Strength Formal Specification Techniques*, April 1995. `http://www.csl.sri.com/sri-csl-fm.html`.
3. J. Denzinger and Dirk Fuchs. Cooperation in theorem proving by loosely coupled heuristics. Technical Report SR-97-04, University of Kaiserslautern, 1997.
4. R.L. Constable et.al. *Implementing Mathematics in the NuPRL Proof Development System.* Prentice–Hall, 1986.
5. A. Felty, D. Howe, and F. Stomp. Protocol verification in Nuprl. In *CAV'98, Lecture Notes on Computer Science.* Springer, 1998.
6. Mark Hayden. The Ensemble system. Technical Report TR98-1662, Cornell University, 1998.
7. Jason Hickey, Nancy Lynch, and Robbert van Renesse. Specifications and proofs for Ensemble layers. In *TACAS '99*, March 1999.
8. Jason J. Hickey. Nuprl-Light: An implementation framework for higher–order logics. In *14th International Conference on Automated Deduction.* Springer, 1997.
9. Paul Bernard Jackson. *Enhancing the NuPRL Proof Development System and Applying it to Computational Abstract Algebra.* PhD thesis, Cornell University, January 1995.
10. Christoph Kreitz, Mark Hayden, and Jason Hickey. A proof environment for the development of group communications systems. In *15th International Conference on Automated Deduction*, pages 317–332. Springer, 1998.
11. Steven P. Miller and Mandayam Srivas. Formal verification of the AAMP5 microprocessor: A case study in the industrial use of formal methods. In *WIFT '95: Workshop on Industrial-Strength Formal Specification Techniques*, pages 2–16, Boca Raton, FL, 1995.
12. Tobias Nipkow and David von Oheimb. Java$_{\ell ight}$ is type-safe — definitely. In *Proc. 25th ACM Symp. Principles of Programming Languages*, pages p. 161–170. ACM Press, New York, 1998.
13. Didier Rémy and Jérôme Vouillon. Objective ML: A simple object–oriented extension of ML. In *ACM Symposium on Principles of Programming Languages*, pages 40–53, 1997.

System Description: Waldmeister – Improvements in Performance and Ease of Use

Thomas Hillenbrand, Andreas Jaeger, and Bernd Löchner

Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany
{hillen,jaeger,loechner}@informatik.uni-kl.de

1 Introduction

Waldmeister is an automated theorem prover for unconditional equational logic. It is based on unfailing Knuth-Bendix completion [1]. During the first stage of development the focus was on efficient rewriting by means of indexing and space saving techniques [2, 4]. In this paper we present two aspects of our recent work which aim at improving the system with respect to performance and ease of use. Section 2 describes a more powerful hypothesis handling. In Sect. 3 we investigate the control of the proof search and outline our current component of self-adaptation to the given proof problem.

2 Enlarging the Hypothesis

A nuisance of completion-based theorem proving is that nearly all the time is spent completing the axiomatization but only a tiny fraction operating on the hypothesis.

The standard approach for tackling a hypothesis is to compute normal forms of the terms and check if they are syntactically equal. Since the rewrite relation is not confluent during the completion, the normal forms are not necessarily unique. Therefore instead of considering an arbitrary normal form, Waldmeister constructs for each term in the hypothesis the set $\mathcal{S}$ of all successors [5]. The hypothesis $u = v$ is proved if $\mathcal{S}(u) \cap \mathcal{S}(v) \neq \emptyset$. During the completion these sets are incrementally expanded.

Our method is best explained with a simple example. Figure 1 shows how sets of successors can be used in solving the proof problem GRP141-1 [6]. The terms $u \equiv f(f(a,b),c)$ and $v \equiv c$ have to be joined. The four graphs illustrate the enlarging of the hypothesis. Initially, the two sets consist of only one term each (a). With the 16th rewrite rule u can be reduced to two different terms which join again in the term denoted by u'. $\mathcal{S}(u)$ now consists of four terms (b). After generating rule no. 19, u can be reduced to v in two different ways, hence $v \in \mathcal{S}(u)$ and the hypothesis is proved (c)! Compare this with the standard approach which would still be stuck in u'. The last graph (d) shows the way out of this dead end which is not possible before 31 rules have been generated. The number of generated rules is cut down by one third here. In larger examples the effects may be far more dramatically.

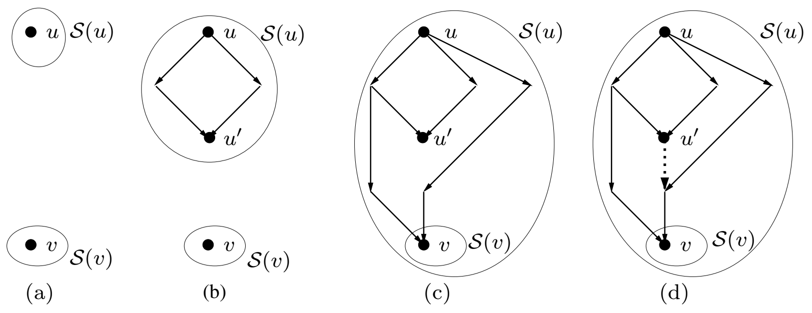


Fig. 1. Sets of successors in proving GRP141-1

The implementation uses one hash table to store the elements of both $\mathcal{S}(u)$ and $\mathcal{S}(v)$. The terms are colored to distinguish whether they descend from u or v. Each time a new rule is generated, all the elements reducible by it are determined, and their rewrite successors added to the hash table. The data structure ensures that every term that is reachable in more than one way, e.g. u' in Fig. 1, appears only once per color. If a term is present in both colors, it is a common successor of u and v.

This technique has improved our system impressively. When introduced, it increased the number of solvable TPTP problems from 278 to 300, and even decreased the average proof time.

The effect of combinatorical explosion occurs frequently with associative and commutative functions and stimulated refinements of the approach. Quite often the sets of successors contain no more than a few hundred elements, but set sizes of 200,000 have been noticed as well. In cases like that we switch back to the standard treatment.

Further enhancements of this more elaborate hypothesis handling include the addition of predecessors as well. This enlarges the sets, and therefore even more care has to be taken of combinatorical explosion.

3 Controlling the Proof Search

The main completion loop is parameterized by a heuristical classification $\mathcal{H}$ and a reduction ordering $>$. While $\mathcal{H}$ can be used to integrate knowledge into the prover [3], Waldmeister currently works with syntactical classifications only. As to $>$, Knuth-Bendix ordering (KBO) and lexicographic path ordering (LPO) are provided.

The influence of distinct classifications and especially orderings is tremendous, often leading to differences in proof times of orders of magnitude. Due to the dynamic nature of the proof process, instantiating these parameters well is a non-trivial task. Some effects are intuitively plausible, whereas others are completely surprising. It seems very difficult, if possible at all, to tackle this task theoretically in sufficient generality.

However, we have experienced that proof problems sharing major parts of their axiomatizations often behave similar in the proof search. This might be explained by two reasons: Firstly, the saturation process is quite independent of the hypothesis to be proved. Secondly, if the axiom sets basically coincide, the saturations will produce many common conclusions.

In order to improve our deduction system's performance, we are following a pragmatic approach. To investigate the properties of a certain domain, we select a representative proof task of moderate difficulty and analyze the influence of different orderings and classifications experimentally. Then the best instantiations of the search parameters are tested on the whole domain. In most cases it is possible to distinguish a setting that is uniformly superior on the whole set of problems.

Of course we cannot expect a potential user of deduction to perform this kind of labor or to supply the prover with a detailed setting on each invocation. Therefore it is sensible to extend the system with a component that automatically adapts it to the problem at hand. This enabled us to integrate our domain-specific knowledge into WALDMEISTER.

3.1 Example: Orderings for Nonassociative Rings

The current TPTP library [6] includes 38 unit equality problems from the domain of nonassociative rings, which differ from the standard noncommutative ones in that the associativity law holds only for triple products where two neighbouring factors are equal [7].

For a series of experiments RNG019-6 was selected. Since we could not find any KBO that solved the problem, we considered the LPO with every possible total operator precedence. With each of these 720 orderings, an attempt to solve the proof task was carried out.

The following can be observed: Despite the fact that 86 % of all orderings failed to prove the problem in reasonable time, there are 57 LPOs with proof times of less than 1 second! The top 30 can be characterized exactly by the precedence constraint $\mathrm{A} \succ \{\cdot, -\} \succ +$ and $\mathrm{C} \succ 0$.

For the generalization step, on all the problems we measured the 30 best orderings vs. 30 randomly chosen from the others. It turned out that the best LPOs got all the problems that could be solved at all, and are exactly those satisfying the precedence constraint $\{\mathrm{A}, \mathrm{C}\} \succ \cdot \succ - \succ +$ and $\mathrm{C} \succ 0$. This is clearly a refinement of the above one, leading towards normal forms which resemble the canonical representation of polynomials, namely as sums of monomials.

Corresponding results were obtained for problems and problem sets in other algebraic domains, e.g. Wajsberg algebras, lattice-ordered groups, or Boolean algebras. Moreover the influence of the search parameter $\mathcal{H}$ was studied in an analogous series of experiments in different domains.

3.2 Automated Control Component

Deriving the instantiation of the search parameters from the set of axioms is a promising way of enhancing a prover's usefulness: Control knowledge is added on the level of algebraic structures to be matched. We use a declarative representation consisting of three compact tables that are easy to edit. The first and the second contain descriptions of usual mathematical concepts, giving the approach a semantical stress, whereas the third factors out the control knowledge.

The process starts with the problem axiomatization as input, e.g. RNG019-6. In the first table algebraic laws, such as associativity or distributivity, are described by means of pattern equations with variables for the operators:

$$\begin{aligned} F(x,F(y,z)) = F(F(x,y),z) \qquad &\Rightarrow \mathrm{Assoc}(F) \\ F(G(x,y),z) = G(F(x,z),F(y,z)) &\Rightarrow \mathrm{Distr_r}(F,G) \end{aligned}$$

The pattern equations are matched onto the axioms, yielding a compact representation of the detected laws: in our case $\{\mathrm{Assoc}(+), \mathrm{Assoc}(\cdot), \mathrm{Distr_r}(\cdot,+), \ldots\}$.

The second table holds algebraic structures characterized as sets of algebraic laws. These sets are matched onto the result of the first stage. For the example RNG019-6 the following two structures match:

$$\begin{aligned} &\{\mathrm{Neut_r}(F,N), \mathrm{Assoc}(F), \mathrm{Inv_r}(F,I,N)\} \Rightarrow \mathrm{Group}(F,I,N) \\ &\{\mathrm{Assoc}(F), \mathrm{Assoc}(G), \mathrm{Distr_r}(G,F), \ldots\} \Rightarrow \mathrm{NonassociativeRing}(F,N,I,G,A,C) \end{aligned}$$

Based on axiom set inclusion, a static conflict resolution mechanism automatically prefers structures that are more specialized, here the ring structure. Finally a parameter instantiation for that structure is looked up in the third table:

$$\begin{aligned} &\mathrm{NonassociativeRing}(F,N,I,G,A,C) \\ &\qquad\qquad \Rightarrow \quad > := \mathrm{LPO}(A \succ C \succ G \succ I \succ F \succ N), \quad \mathcal{H} := \text{term size} \end{aligned}$$

Not only is the representation lucid and easy to maintain and extend, but also is this staged matching more efficient in practice than the direct one.

Table 1 shows the results of an experimental comparison of this self-adapting control with our standard general-purpose strategy. In both settings, Waldmeister was run on a Sun SPARCstation 20/712 (75 Mhz, 192 MB main memory). The test set consisted of all TPTP unit equality problems. We measured the performance with respect to the number of problems solved within a time limit of 10 minutes. The results are given for the whole test set as well as for the subsets formed by the larger TPTP domains.

The new approach currently increases the percentage of the solvable share from 82 to 87 %. As not shown here, it also cuts down the average proof time from 17 to 6 seconds. Further progress may be expected with the integration of more control knowledge.

Table 1. Experimental comparison of control approaches

Problem set	Size	general purpose	self-adapting
TPTP	425	82 %	87 %
RNG	56	32 %	52 %
COL	117	86 %	93 %
BOO	42	98 %	100 %
GRP	133	95 %	96 %

4 Availability

Guided by the needs of students attending local lectures, we have just realized a convenient Web-interface for WALDMEISTER that is reachable via:

`http://www-avenhaus.informatik.uni-kl.de/waldmeister`

From this URL further documents and the complete distribution containing an easy-to-use version together with problems and a manual are available.

References

[1] L. Bachmair, N. Dershowitz, and D.A. Plaisted. Completion without failure. In *Resolution of Equations in Algebraic Structures*, volume 2, pages 1–30. Academic Press, 1989.

[2] A. Buch and Th. Hillenbrand. WALDMEISTER: development of a high performance completion-based theorem prover. SEKI-Report 96-01, Universität Kaiserslautern, 1996.

[3] J. Denzinger and S. Schulz. Learning domain knowledge to improve theorem proving. In *Proceedings of the 13th International Conference on Automated Deduction*, volume 1104 of *LNAI*, pages 76–62. Springer Verlag, 1996.

[4] Th. Hillenbrand, A. Buch, R. Vogt, and B. Löchner. Waldmeister: High performance equational deduction. *Journal of Automated Reasoning*, 18(2):265–270, 1997.

[5] A. Jaeger. Anwendungen von Nachfolgermengen in Termersetzungssystemen. Projektarbeit, Universität Kaiserslautern, 1997. In German.

[6] C.B. Suttner and G. Sutcliffe. The TPTP problem library (TPTP v2.2.0). Technical Report 99/02, Department of Computer Science, James Cook University, Townsville, Australia, 1999.

[7] K.A. Zhevlakov, A.M. Slin'ko, I.P. Shestakov, and A.I. Shirshov. *Rings that are nearly associative.* Academic Press, New York, 1982. Translated from the Russian by H.F. Smith.

Formal Metatheory Using Implicit Syntax, and an Application to Data Abstraction for Asynchronous Systems

Amy P. Felty[1], Douglas J. Howe[1], and Abhik Roychoudhury[2]

[1] Bell Labs, Murray Hill, NJ 07974, USA
`{felty,howe}@bell-labs.com`
[2] Dept. of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11790, USA
`abhik@cs.sunysb.edu`

Abstract. Abstraction is a useful tool in verification, often allowing the proof of correctness of a large and complex system to be reduced to showing the correctness of a much smaller simpler system. We use the Nuprl theorem prover to verify the correctness of a simple but commonly occurring abstraction. From the formal proof, we extract a program that succeeds when the abstraction method is applicable to the concrete input specification and in this case, computes the abstracted system specification. One of the main novelties of our work is our "implicit syntax" approach to formal metatheory of programming languages. Our proof relies entirely on semantic reasoning, and thus avoids the complications that often arise when formally reasoning about syntax. The semantic reasoning contains an implicit construction of the result using inductive predicates over semantic domains that express representability in a particular protocol language. This implicit construction is what allows the synthesis of a program that transforms a concrete specification to an abstract one via recursion on syntax.

1 Introduction

Theorem proving and model checking can be usefully combined by using a theorem prover to verify abstractions of protocols or system specifications. In particular, one can often use a model checker to verify some property of a protocol that has an infinite or intractably large state space, by first transforming the original or *concrete* protocol into a more *abstract* version for which model checking is feasible [13,2]. A theorem prover can be used to check, for example, that the property (or some transformation of it) holds of the abstract protocol if and only if it holds of the original protocol. This can be done directly by formalizing the two versions of the protocol and proving the specific property of interest. This approach is taken in [7], for example, using the integration of a BDD based model checker as a decision procedure in PVS [11]. One can also, as in [10], provide general support for doing this kind of reasoning by formalizing a refinement calculus and methodology relating system specifications and abstractions; or as in [5], use a model checker with assumption commitment style reasoning on the abstract system and then use a theorem prover to discharge the assumptions in the concrete system.

Typically, when a system specification is represented in a theorem prover, a so called “shallow embedding” is used. In a shallow embedding of a programming or specification language in a theorem prover, programs and specifications are directly interpreted in the logic of the theorem prover. Thus, one formalizes only the semantics of the language. For example, the commands of an imperative programming language might be encoded as objects of type $com = state \rightarrow state$.

In contrast with shallow embedding is “deep embedding”, where both the semantics *and* syntax of the embedded formalism are explicitly represented in the theorem prover. Using this approach, one might have a type *comsyn* consisting of abstract syntax trees of commands, and then a meaning function $M \in comsyn \rightarrow com$. Deep embeddings are considerably more difficult to reason about in theorem provers. In practice, shallow embeddings are used whenever possible, and deep embeddings are done only when one is interested in some property that cannot be expressed by referring to semantic objects alone. A comparison of these two methods is presented in [1], for example.

In this paper we show how to exploit the constructivity of the Nuprl theorem prover [4] to synthesize a particular verified-correct abstraction algorithm. We build a proof in Nuprl from which we can extract a program that takes a concrete specification as input, tests whether the abstraction method applies to it, and if so, returns the abstracted system specification.

One of the main novelties here is that we do *not* use a deep embedding. Our proof reasons only about semantics, yet we are able to synthesize a program that operates on syntax. Thus we reason only about the semantic aspects of the abstraction method, even though we are implicitly constructing the program that builds abstracted programs. The central idea is to define inductive predicates over semantic domains that express representability in a particular protocol language. We give a small illustrative example of the approach in Section 2.

It is not obvious that this notion of representability is adequate for non-trivial examples. Many concepts that are natural in reasoning about syntax cannot be directly expressed. For example, we cannot directly write down a function which takes a command and returns a list of all program variables occurring in it, since a command is just a function on states that is assumed to be representable, and we cannot in general determine the list of variables from this function.

As evidence for the practicality of our approach, we apply it to a simple but common *data abstraction* method. The correctness of the abstraction, as well as the representability of the abstract system specification, was proved in Nuprl. We used the program extracted from these proofs to obtain the abstraction of a simple communication protocol.

The only other paper we know of that uses the idea of representing syntax implicitly in type theory via an inductive predicate is [3], where it is proposed as a way of defining internal computational complexity measures. Nothing was implemented, and no proofs are given. Furthermore, the paper does not address the use of implicit syntax together with extraction to synthesize metaprograms.

We are aware of one other effort involving program extraction and model checking [12], in which the correctness proof of a model checker in the Coq proof checker yields, via extraction, an executable model checker which is then considered as a trusted decision procedure.

2 Example

Before proceeding to our data-abstraction case study in Nuprl, we illustrate our approach with a simple, rather artificial, example involving a trivial imperative programming language P where programs are sequences of assignments of variables to variables. The example is presented at the level of constructive mathematics, and has not been implemented in Nuprl.

We start with a semantic account of the language. We represent variable names as strings and assume that variables take on integer values; we define $\mathit{state} = \mathit{string} \to Z$, and define the type of (meanings of) commands to be $\mathit{com} = \mathit{state} \to \mathit{state}$. Assignment and command sequencing can now be defined semantically:

$$\begin{aligned} \mathit{assg} : \mathit{string} \to \mathit{string} \to \mathit{com} &= \lambda x.\, \lambda y.\, \lambda s.\, s[x \leftarrow s(y)] \\ \mathit{seq} : \mathit{com} \to \mathit{com} \to \mathit{com} &= \lambda c_1.\, \lambda c_2.\, \lambda s.\, c_2(c_1(s)) \end{aligned}$$

where $s[x \leftarrow l]$ is the state which maps x to l and all other variables y to $s(y)$. Write $x := y$ for $\mathit{assg}(x)(y)$ and $c_1; c_2$ for $\mathit{seq}(c_1)(c_2)$.

Consider the following fact about P: for every command c, if there is a variable x such that c only affects the value of x, then c is equivalent to a single assignment statement. This fact is false in general for members of com; to formalize it, we need to somehow reason about the syntax of P. We do this by inductively defining a representability predicate $R : \mathit{com} \to P_1$ as follows (where P_1 is the type of Nuprl propositions).

$$\begin{aligned} R(c) \Leftrightarrow\ & \exists\, x, y \in \mathit{string}.\ c = (x := y) \\ & \vee\ \exists\, c_1, c_2 \in \mathit{com}.\ R(c_1) \wedge R(c_2) \wedge c = (c_1; c_2) \end{aligned}$$

Define $u(c)$, for $c \in \mathit{com}$, if there exists $x \in \mathit{string}$ such that for all $y \neq x \in \mathit{string}$ and all $s \in \mathit{state}$, $s(y) = c(s)(y)$. Our fact may now be formalized as follows.

$$\forall\, c \in \mathit{com}.\ R(c) \Rightarrow u(c) \Rightarrow \exists\, x, y \in \mathit{string}.\ c = (x := y)$$

Unfortunately, the obvious proof attempt, using induction on the definition of $R(c)$, fails, because for the case when $c = (c_1; c_2)$, we get to a point where we need to show that $u(c)$ implies $u(c_1)$ and $u(c_2)$, and this is not necessarily true.

If we strengthen the assumption on the representation of c to require that u hold of all subcommands, then this obvious proof will work. We can state this property by modifying the definition of representability. In particular, define $R_q(c)$, for q a predicate on com, by replacing $R(c)$ by $R_q(c)$ in the definition of $R(c)$ and conjoining the right-hand side of the definition with $q(c)$. We can prove

$$\forall\, c \in \mathit{com}.\ R_u(c) \Rightarrow \exists\, x, y \in \mathit{string}.\ c = (x := y)$$

by a straightforward induction on $R_u(c)$. Since we are formalizing in a constructive logic, the proof will yield a program that takes a c and evidence for $R_u(c)$ and produces the x, y such that $c = (x := y)$.

In order to run this program on a particular c, we need a proof of $R_u(c)$. Since there is no general method in the type theory to go from a member of *com* to a representation, we assume we are given a proof of $R(c)$. The method for going from c to $R(c)$ can be implemented in Nuprl's metalanguage. The problem is now to go from $R(c)$ to $R_u(c)$. This is not always possible, so we choose, as a simple sufficient condition, to do this only for commands whose constituent assignments have a unique variable on the left hand side.

To deal with this kind of syntactic sufficient condition, we define a *possibility* operator on propositions, denoted ?. In particular, the formula $?A$ is defined to be $A \vee True$. Clearly, for any A, $?A$ holds because the right disjunct is provable. When we prove a theorem whose conclusion is $?A$, we take care to choose to prove the left disjunct (A) in situations where the sufficient condition is known to hold. Theorems of this form give a partial correctness result. The program that Nuprl extracts from the proof will either return a result of type A or the constant *axiom* which is the proof of $True$. The fact that it does not return the trivial result *axiom* when the sufficient condition holds is purely a metatheoretic one.

Returning to the example, we can prove a theorem $\forall c \in com.\ R(c) \Rightarrow ?R_u(c)$ by induction on the definition of $R(c)$. If we construct the right kind of proof, the extracted program will translate evidence for $R(c)$ into evidence for $R_u(c)$ in the case where c satisfies the sufficient condition given above.

3 Protocol Verification in Nuprl

For our data-abstraction case study, we use the environment for protocol verification that was built in the course of verifying the SCI cache coherence protocol [6]. Here, we briefly describe our shallow embedding of a Unity-like guarded command language in which protocols are expressed. We illustrate this language with our running example presented in Fig. 1. In this language, a *specification* or *program* is a list of guarded *actions*, each having a *guard* and a *body*, along with an initial condition on values of program variables. In general guards can be message receives or boolean conditions, and bodies can contain assignments, conditionals, and message sends. The example presents a protocol with three distinct processes—Sender, Channel (or Queue), and Receiver, denoted s, q, and r, respectively. Consider the two actions for the Receiver, marked r_1 and r_2. In r_1 the guard always holds and the body contains a message to the sender requesting data, where *request* is the *message type*. A message can also contain arguments as illustrated in r_2. Here, the guard indicates that this action can be executed if the first message in *buf*[r] (r's message buffer) has type *rsend*. This message has one argument, *rdata*, containing the requested data. The message is removed from the queue (received) and the body is executed. The Sender and Channel processes both have a single parametrized action (s_d and q_m re-

```
Program  $P$:
  Initial Condition: $sent = -1$
  Actions: $[r_1, r_2, s_0, \ldots, s_{n-1}, q_m]$ for some $n > 0, m \geq 0$

  $(r_1)$ $true \longrightarrow s!request$
  $(r_2)$ $buf[r]?rsend(rdata) \longrightarrow rcvd := rdata$

  $(s_d)$ $buf[s]?request \longrightarrow sent := d; q!qsend(sent, 0)$

  $(q_m)$ $buf[q]?qsend(data, i) \longrightarrow$
          if $i = m$ then $r!rsend(data)$ else $q!qsend(data, i + 1)$ fi

Property $\psi$:
  Data delivery: $\forall y.\ G(sent = y \Rightarrow F(rcvd = y))$
  Order preservation: $\forall y_1.\ \forall y_2.\ G((sent = y_1 \wedge F(sent = y_2)) \Rightarrow$
                                       $F(rcvd = y_1 \wedge F(rcvd = y_2)))$
```

Fig. 1. Running Example

spectively). Action s_d is parametrized by the value of the data transmitted to the Channel. Action q_m is parametrized by the length of the channel. Thus, the above specification represents a collection of programs where the number of data values and the length of the queue are bounded by an arbitrary finite number. We prove universally quantified linear time temporal logic properties (such as the data delivery property in Fig. 1) of the example protocol, by performing data abstraction of the protocol.

Nuprl is a goal-directed interactive theorem prover in the style of LCF. It implements a constructive type theory with a rich set of constructors. Because of the constructivity, programs can be extracted from proofs. Logic is encoded via the propositions-as-types principle, whereby a proposition is identified with the type of data that provides evidence for the proposition's truth. The version of Nuprl we use [8] also supports classical reasoning, which can be used in any part of a proof that does not affect the extracted program. Formal mathematics in Nuprl is organized in a single library, which is broken into files simulating a theory structure. Library objects can be definitions, display forms, theorems, comments or objects containing ML code. Definitions define new operators in terms of existing Nuprl terms and previously defined operators. Display forms provide notations for defined and primitive operators. Theorems have tree structured proofs, possibly incomplete. Each node has a sequent, and represents an inference step. The step is justified either by a primitive rule, or by a *tactic*. Tactics provide automation to help with goal-directed search.

Our embedding of the semantics of state transition systems in Nuprl is fairly straightforward. We define a state as a pair where the first component is the usual mapping from identifiers to values (integers). The second component is a *history* variable that records the sequence of messages that have been sent and received during the entire execution. This history variable is important for reasoning about data that passes via messages. Messages have two components. Message types such as *rsend* are encoded as integers as the first component of a message. The second component is a list of integers that encodes the message's arguments.

Expressions and commands are defined as functions on state. Define *exp* to be $state \rightarrow Z$, and, as in Section 2, define *com* to be $state \rightarrow state$. The commands and expressions used in Figure 1 are defined as functions over these types, and Nuprl's display forms are used to give their applications conventional notations. We use a dot notation for the value of a command or expression in a state, such as $e \cdot s$ for (es).

A program is defined as a pair containing a list of commands and an initial condition which is a predicate on state (of type $state \rightarrow P_1$ where P_1 is the type of Nuprl propositions). The initial condition must at least require that the history start out empty. In our model, a command is enabled if it changes the state when applied. Thus commands whose guards are true but do not change the state are considered disabled. A trace is defined in the usual way as a function of type $N \rightarrow state$. A predicate *trace_of* encodes the restriction that for any n, there is an action such that when applied to state n results in state $n+1$. Temporal operators such as G (always) and F (eventually) are defined as predicates on traces (of type $trace \rightarrow P_1$) using a fairly direct encoding of the definitions in [9]. We then define the notion of a property ψ being valid of a program P in the usual way as $P \models \psi$ iff $\forall tr : trace.trace\_of(P, tr) \rightarrow \psi(tr)$. In [6], the automation that we developed for the verification of protocols in Nuprl was discussed in detail. In the present work, we draw mainly on the machinery for rewriting, which draws on a large body of equality theorems for protocols.

4 Overview of Data Abstraction

In this section, we give an overview of the form of data abstraction used in our case study. Suppose we are given a program P and a property ψ of traces of the program, and we want to verify whether $P \models \psi$, *i.e.* whether all traces of P satisfy ψ. Suppose P contains a variable v that can take on an arbitrarily large number of data-values. We may be able to perform "data-value abstraction" on v to create an abstract program P' and an abstract property ψ' such that $P' \models \psi' \Leftrightarrow P \models \psi$ and such that v takes on values from a smaller set during execution of P'.

We first discuss how to compute an abstract program from a concrete program, and then discuss some sufficient conditions, that can be checked statically, under which this abstraction is safe.

In our example program in Fig. 1, the data that we are particularly interested in and whose values we want to abstract is the value that gets assigned to the identifier *sent*. The flow of this value through the program execution is important for proving both the data delivery property and the order preservation property mentioned in the figure. We formalize this flow as a set of identifiers that are affected by the value of *sent*. We must also consider communication via message buffers. To take this into account, we define a *message reference* to be a pair $\langle T, i\rangle$ where T is a message type, and i is natural number denoting a position in the list of arguments to a message. A *data reference* of a program is either a program variable or a message reference. For example, the value of *sent* gets passed via the message reference $\langle \mathit{qsend}, \mathit{0}\rangle$.

From now on, we will use $\mathbf{d}$ to denote the set of all data references possibly affected by the values of the variable(s) being abstracted. In our example, we have

$$\mathbf{d} = \{\mathit{sent}, \langle \mathit{qsend}, 0\rangle, \mathit{data}, \langle \mathit{rsend}, 0\rangle, \mathit{rdata}, \mathit{rcvd}\}.$$

Clearly it is often possible to compute a suitable $\mathbf{d}$, but we have not done this in our case study and so we do not elaborate on it.

Our Nuprl development is parameterized with respect to the *abstraction function* φ, also called a *collapsing function*, that will map the values taken on by the data references in $\mathbf{d}$ to a small finite domain.

In our running example, in order to verify the data delivery property in Fig. 1 we will abstract the data values to a two valued abstract domain. For instance, we can use the functions φ_y, parameterized by the y of our data delivery property, defined by $\varphi_y(n) := (\mathit{if}\ n = y\ \mathit{then}\ y\ \mathit{else}\ {-1})$. Using this function corresponds to tracking the delivery of the data-value y. The value -1 represents all other concrete values.

The abstraction function φ_y is parameterized by y, so we would need to generate a new abstract program for every y of interest. However, note that in our example, the only processes to assign to *sent* are the processes s_d, and that the possible assigned values are $\{0 \ldots n-1\}$. The protocol is symmetric on these values: if we apply a permutation of theses values to the right-hand-sides of all assignments of constants to *sent*, then we get the same protocol. Because of this symmetry, checking the data-delivery property for an arbitrary y in $\{0 \ldots n-1\}$ is the same as checking it for $y = 0$.

We can compute the abstract program P', given φ, as follows. First compute $\mathbf{d}$, and then, for all $u \in \mathbf{d}$, replace any constraint $u = n$ in the initial condition of P, where n is a constant, by $u = \varphi(n)$, and replace any assignment $u := \mathit{expr}$ (and any check $u = \mathit{expr}$) in any action of P by $u := \varphi(\mathit{expr})$ $(u = \varphi(\mathit{expr}))$. If we know that data values in $\mathbf{d}$ are only passed around and not manipulated (and if we know that the property ψ we want to verify satisfies certain properties) then we are guaranteed that our data-value abstraction preserves enough information to verify property ψ.

We make this abstraction method more precise as follows. Suppose that: $D = \{0 \ldots d-1\}$, $m \in D$, $\mathbf{d}$ is a set of data references, $\varphi \in Z \to D$, P and P' are programs, and ψ is a predicate on traces. We first describe how to lift φ (the abstraction function) to states and traces. In particular, we define a function on states, denoted $\gamma_{\mathbf{d}}^{\varphi}$, where φ is the function to be lifted and $\mathbf{d}$ is a set of data references. The collapsed state $\gamma_{\mathbf{d}}^{\varphi}(s)$ is obtained from state s by mapping the value of each program variable x in $\mathbf{d}$ to $\varphi(x \cdot s)$ where $\varphi(x \cdot s)$ denotes the value of x in state s, and applying φ to each value t such that there is a message reference $\langle T, i\rangle$ in $\mathbf{d}$ and t is the i^{th} argument to a message of type T in the history component of s. We will often just write γ when φ and $\mathbf{d}$ are obvious from context. Traces being infinite sequences of states, we define $(\gamma_{\mathbf{d}}^{\varphi}(tr)).i = \gamma_{\mathbf{d}}^{\varphi}(tr.i)$ where $tr.i$ denotes the i^{th} state in the trace tr, for any natural number i. Note that we overload $\gamma_{\mathbf{d}}^{\varphi}$. Let $trace(P)$ denote the set of all valid traces of program P.

Conditions for P' to be a correct abstraction of P are as follows.

1. $\forall tr.(tr \in trace(P)) \Leftrightarrow (\gamma_{\mathbf{d}}^{\varphi}(tr) \in trace(P'))$
2. $\forall tr.\ \gamma_{\mathbf{d}}^{\varphi}(tr) \models \psi(m) \Leftrightarrow tr \models \psi(m)$
3. For all permutations f of D, $\forall tr.(tr \in trace(P)) \Leftrightarrow (\gamma_{\mathbf{d}}^{f}(tr) \in trace(P))$
4. For all permutations f of D, and for all $k \in D$, $\forall tr.\ tr \models \psi(k) \Leftrightarrow \gamma_{\mathbf{d}}^{f}(tr) \models \psi(f(k))$

If the above conditions hold true then $P' \models \psi(m)$ iff $P \models \forall y{\in}D.\ \psi(y)$.

Our Nuprl proof captures sufficient conditions for (1) and (3) to hold. The extracted program will check that these conditions hold for a given P (P' will be a function of P). We have not formalized the syntax of temporal logic, so our program does not check any sufficient conditions for ψ. In particular, conditions (2) and (4) are proved by hand on a case-by-case basis.

A generic sufficient condition for condition (1) is that the control flow of program P is completely independent of the values of the data references in $\mathbf{d}$. For example, there can be no conditional branching on the value of variables in $\mathbf{d}$. Additionally, the initial condition of the program and the guards of the program actions must be independent of the values of the data references in $\mathbf{d}$.

A sufficient condition for (3) is similar to the one for (1), except that we additionally require that in any action a containing an assignment $x := n$, where $n \in D$ and x is in $\mathbf{d}$, all assignments in a of constants to members of $\mathbf{d}$ have n as the right-hand side, and, furthermore, for every other $k \in D$, there is another action a' such that a' is the result of replacing in a each assignment of the form $z := n$ by $z := k$.

5 Defining Representability in Nuprl

In this section, we define representability of commands which, as mentioned, allows us to reason semantically about data abstraction, while implicitly constructing a program that operates on syntax. In the interests of compactness, in this section, as well as later sections, we will usually use a more mathematical style of presentation instead of giving exactly what would would appear in the theorem prover. The differences in presentation are minor notational ones.

To talk about the representability of commands, we also need to define the representability of expressions. In both cases, we parameterize by a state invariant, since ultimately we will only want a program and its representation to be equivalent on certain states. In our case, we only need to consider states collapsed by γ. We define equality up to invariant I of functions on states (such as expressions and commands), written as $=_I$, as equality of values on all states satisfying I.

Representability of expressions, denoted $R_I(e)$ or $R[I](e)$, is inductively defined below. We omit the types of bound variables when they are clear from context. $R_I(e)$ is true iff

$$\begin{aligned}
&e =_I \mathit{false} \vee e =_I \mathit{true} \vee [\exists n : \mathcal{Z}.e =_I n] \vee [\exists x : id.e =_I x] \\
&\vee [\exists b, e_1, e_2.(R_I(b) \wedge R_I(e_1) \wedge R_I(e_2)) \wedge e =_I (\mathit{if}\ b\ \mathit{then}\ e_1\ \mathit{else}\ e_2)] \\
&\vee [\exists e_1, e_2.R_I(e_1) \wedge R_I(e_2) \wedge (e =_I (e_1 + e_2) \vee e =_I (e_1 - e_2) \vee e =_I (e_1 = e_2) \\
&\qquad\qquad\qquad\qquad\qquad \vee e =_I (e_1 \vee e_2) \vee e =_I (e_1 \wedge e_2))] \\
&\vee [\exists e'.R_I(e') \wedge e =_I \neg e']
\end{aligned}$$

We use several abbreviations here. For example, n in the equality $e =_I n$ denotes $\lambda s.n$ and x in the equality $e =_I x$ denotes $\lambda s.(x \cdot s)$. Note that we overload the operators in binary expressions. For example $\wedge$ also denotes the conjunction of Nuprl.

Representability of commands is parameterized by an invariant, as above, and also by a predicate on commands. Intuitively, $R_{I,Q}(c)$ (also denoted $R[I,Q](c)$) means that c is representable, up to I, in such a way that for each subcommand c', Q is true and c' preserves I. The exact right-hand side in Nuprl of the definition of $R_{I,Q}(c)$ (denoted `rcom[I,Q]` in Nuprl) is the following.

```
(c =[I] skip
∨ (∃x:id. ∃e:zexp. rexp[I] e ∧| c =[I] x:=e)
∨ (∃c1,c2:com. (rcom[I,Q] c1 ∧ rcom[I,Q] c2) ∧| c =[I] (c1 ;c2))
∨ (∃b:zexp ∃c1,c2:com
     (rexp[I] b ∧ rcom[I,Q] c1 ∧ rcom[I,Q] c2)
     ∧| c =[I] (if b then c1 else c2))
∨ (∃b:zexp. ∃c':com. (rexp[I] b ∧ rcom[I,Q] c') ∧| c =[I] b --> c')
∨ (∃p:PId ∃d:zexp ∃M:Z. ∃as:zexp List.
     (rexp[I] d ∧ ∀(rexp[I];as)) ∧| c =[I] d!M(as))
∨ (∃p:PId. ∃c':com. ∃M:Z. ∃as:id List.
     rcom[I,Q] c' ∧| c =[I] p?M(as) --> c'))
∧| (Q c ∧ (∀s:state. I s ⇒ I (c s)))
```

The occurrence of $\forall$ applied to two arguments has the meaning that the property (the first argument) holds of every element of the list (the second argument). The operator ∧| is an alternate definition of conjunction in Nuprl which roughly makes the right hand side computationally insignificant, so that an extracted program producing a witness for the conjunction will only produce witnessing information for the left hand side. Using such alternate definitions can dramatically improve the computational efficiency of extracted programs.

For representability of programs, in addition to commands, we must represent the initial condition predicate. We choose to represent it as a command that only sets variable values. The initial states are those that result from running this command on a state with an empty history. We overload R again and use $R_{I,Q}(P)$ and $R[I,Q](P)$ to denote representability of programs. A program is representable if the initial state command is representable and each of the actions are representable. We omit its precise definition.

6 Main Results of the Nuprl Formalization

In this section, we discuss the culminating theorems of our formal proof development in Nuprl and illustrate how the program we extract from the formal proofs computes a data-abstracted version of a concrete program as long as the concrete program satisfies the condition that the control flow is independent of the data-values. We first give some additional definitions.

Instead of stating control/data independence explicitly as a requirement on programs, we will prove the theorems in such a way that the extracted program is a partial function that will succeed if the condition is satisfied and will fail otherwise. To do so, we use the possibility operator defined in Section 2.

In addition to lifting φ to states and traces as in Section 4, we also lift it to commands and programs. For commands, we have $\gamma_{\mathbf{d}}^{\varphi}(c) = (\gamma_{\mathbf{d}}^{\varphi} \circ c)$. Thus, applying a collapsed command is the same as applying a command to a state and then collapsing the state. For a program $P \equiv \langle as, I\rangle$ (as is the list of commands, I denotes the initial condition), we have

$$\gamma_{\mathbf{d}}^{\varphi}\langle as, I\rangle = \langle map\ (\gamma_{\mathbf{d}}^{\varphi})\ as,\ \lambda s.\exists s'.(s = \gamma_{\mathbf{d}}^{\varphi}(s') \wedge I(s'))\rangle$$

where *map* is the usual mapping function on lists, and the first occurrence of $\gamma_{\mathbf{d}}^{\varphi}$ on the right hand side denotes the collapsing function for commands, while the second denotes the collapsing function for states. The function $\gamma_{\mathbf{d}}^{\varphi}$ on programs gives us a semantic notion of abstract program, which we call the *pseudo-abstract program.*

There are two main theorems. The first of these is

```
∀drs:dref List. ∀phi:(idempotent).
∀p:prog. ∀psi: { f:trace → P | respects(f;γ[drs;phi]) }.
  rprog p
  ⇒ (∀e:zexp. rexp e ⇒ rexp (phi o e))
  ⇒ ?(rprog[im(γ[drs;phi]),·] (γ[drs;phi] p)
        ∧| (p |= psi ⟺ γ[drs;phi] p |= psi))
```

This theorem says that given a list of data references, an idempotent collapsing function on integers, a representable program, and a temporal property satisfying a certain condition, then possibly the pseudo-abstract program is representable (up to states in the image of the abstraction function) and is equivalent with respect to the property `psi`. The idempotence requirement is a technical detail that is explained later. We have defined specialized display forms for some operators in Nuprl, so, for example, `rprog[I,Q]` displays as just `rprog` in the case that both `I` and `Q` are `λx. True`.

The second theorem is

```
∀drs:dref List. ∀d:N. ∀p:prog.
  rprog p
  ⇒ (∀psi: { f:Z → trace → P | perm_inv(d;drs;f) }
       ?( ↓( ∀y0:Nd. (∀y:Nd. p |= psi y) ⟺ p |= psi y0) ) )
```

This theorem says that if `psi` is a function from integers to temporal properties satisfying a certain permutation invariance property, then possibly for all `y0` in the set $\{0, \ldots, d-1\}$, the program satisfies `psi` at all y iff it satisfies it at `y0`.

We apply our abstraction method to a particular program P (which we assume has been entered into Nuprl as a member of type `prog`) and to a particular function ψ from integers to temporal properties, by doing the following.

1. Prove a theorem that P is representable.
2. Prove that ψ satisfies the condition in the second theorem above, and that $\psi(0)$ satisfies the condition in the first theorem.
3. Run the extraction of the second theorem with arguments **d**, some natural number d, P and the extracted program from the representability theorem. If the result is of the form *inl*(.), then the property under the `?` holds and so it suffices to check the program satisfies $\psi(0)$; otherwise halt.
4. Run the extract from the first theorem on appropriate inputs. If the result is of the form $inl(x)$, then x will encode a representation of the abstracted program.

Part (1) has been automated. Part (2) is manual and corresponds to parts (2) and (4) of Section 4. In part (3), the list **d** must be entered manually. It would be straightforward to write an ML function to compute such a list given P, but we have not done this. We have implemented a procedure that takes the encoding produced by step (4) and makes it readable. We have proven theorems which condense some of the steps above in minor ways, but we believe the above account is clearer. We have not bothered implementing uninteresting procedures to take ascii representations of protocols to Nuprl representations, nor to completely glue together the steps above.

We have applied our method to an instance of the program in Fig. 1. Let P_0 be the instance with the 3 data values $\{0, 1, 2\}$ and a queue of length 8. We choose the function φ_0, take $d = 3$, and take $\psi(y)$ to be $G(sent = y \Rightarrow F(rcvd = y))$. We also use the six-element set given earlier as as the set **d** of data references. Applying the steps above succeeds, and yields the result below. We use the following abbreviation where e is any expression: $F(e) :=$ (**if** $e = 0$ **then** 0 **else** -1). The term $s'.2$ denotes the history component of the state.

Initial Condition:

$$\lambda s.\exists s'[(s'.2 = nil) \wedge s = (sent := -1; sent := F(sent);$$
$$data := F(data); rdata := F(rdata); rcvd := F(rcvd); skip)(s')]$$

(r_1) $true \longrightarrow s!request$
(r_2) $buf[r]?rsend(rdata) \longrightarrow rcvd := (F(rdata)$
(s_2) $buf[s]?request \longrightarrow sent := F(2); q!qsend(F(sent), 0)$
(s_1) $buf[s]?request \longrightarrow sent := F(1); q!qsend(F(sent), 0)$
(s_0) $buf[s]?request \longrightarrow sent := F(0); q!qsend(F(sent), 0)$
(q_8) $buf[q]?qsend(data, i) \longrightarrow$
if $i = 8$ **then** $r!rsend(F(data))$ **else** $i := i + 1; q!qsend(F(data), i)$ **fi**

Because the steps succeeded, it is guaranteed that checking $\forall y \in \{0 \ldots 2\}.\psi(y)$ holds for P_0 is equivalent to checking that $\psi(0)$ holds for the above program. Note that some trivial simplifications are possible, for example reducing or eliminating some applications of F, and collapsing the identical actions s_1 and s_2. These simplifications would be straightforward to implement, but we have not done so yet.

We wrote a small Nuprl program, given below, to glue together the computational parts for this example. The evaluator for Nuprl programs is a basic call-by-need interpreter, and so is quite slow. We implemented a simple-minded general program optimizer before running the example program. The example terminated in about 5 seconds (on a 400MhZ PC).

Below is a closed Nuprl term whose evaluation produces the representation of the abstracted program.

```
let phi = phi_eg(0) in let psi = λy.psi_eg1(y) in
let p = sqr_inst1 in let p_rep = ext{sqr1_rep} in
let drs = sqr_drs1 in let phi_rep = ext{phi_eg_rep} 0 in let d = 3 in
if isl(ext{poss_data_indep} drs d p p_rep psi)
then let res = ext{abs_thm_2} drs phi p (psi 0) p_rep phi_rep in
     if isl(res) then outl(res) else "No" fi
else "No" fi
```

The expressions `ext{abs_thm_2}` and `ext{poss_data_indep}` name the respective programs extracted from the two theorems discussed above. Recall that both of these programs produce a value in a disjoint union. The program above first uses `poss_data_indep` to test if the example program (bound to `p`, with representation bound to `p_rep`) satisfies the permutation invariance property expressed in the second theorem. If so (*i.e.* if the result is in the left part of the disjoint union), then it runs `abs_thm_2`, testing the result for success using `isl`. In the unsuccessful cases, `"No"` is returned.

The output of this program is an explicit piece of data that completely specifies the required abstract program. However, it is rather hard to read, involving numerous injections into disjoint sums, and also junk such as parts of the program's semantics. To help with readability, we implemented a conventional kind of recursive data type in the type theory for representing terms and expressions, and extracted a function that translates to this second representation. From this latter representation, we obtained by inspection the form of the abstracted program given above.

7 Some Details of the Nuprl Proofs

Most of the work in the proof is related to conditions (1) and (3) of Section 4 and is independent of the kind of temporal properties being checked. We give details only on the parts related to condition (1). Most of the work related to condition (3) is similar. The work related to condition (1) is divided into three main theorems. These theorems form the bulk of the proof of the first "top-level" theorem given Section 6. We discuss each theorem below, and describe a few example steps of their proofs. In what follows, we are taking the P' in condition (1) to be the pseudo-abstract program $\gamma^{\varphi}_{\mathbf{d}}(P)$.

For all three theorems, we assume that P is some program, $\mathbf{d}$ a set of data references, and φ an idempotent function on integers. The idempotence requirement is necessary to show that certain commands satisfy the homomorphism property discussed below. In discussing the theorems, we omit the subscript and superscript on occurrences of $\gamma_{\mathbf{d}}^{\varphi}$.

The first theorem says that condition (1) holds assuming that for all commands c in P, $(\gamma \circ c) = (\gamma \circ (c \circ \gamma))$. We call this latter property the *homomorphism property* on commands, and denote it as $hom_{\gamma}(c)$. The reason that we consider this property is that it is a simple semantic sufficient condition for the control flow of P to be independent of the data references in $\mathbf{d}$.

Let ψ be a temporal property, *i.e.* a predicate on traces. Define $respects(\psi, F)$ to be the proposition $\forall tr.\psi(tr) \Leftrightarrow \psi(F(tr))$. Our first theorem is the following.

Theorem 1. *Suppose $R[True, hom_{\gamma}](P)$ holds, and let ψ be a temporal property such that $respects(\psi, \gamma)$. Then $P \models \psi \Leftrightarrow \gamma(P) \models \psi$.*

The second theorem embodies a check of a syntactic sufficient condition for the condition $R[True, hom_{\gamma}](P)$ of Theorem 1.

Theorem 2. *If $R[True, True](P)$ then $?R[True, hom_{\gamma}](P)$.*

We prove this theorem by induction on representability, considering a case $?R[True, hom_{\gamma}](c)$ for each type of command c, and then proving the property $R[True, hom_{\gamma}](c)$ for the commands where it can be seen to hold according to our sufficient condition. If we were using an explicit approach to syntax, this inductive argument would correspond to a proof by induction over syntax trees that (possibly) the homomorphism property holds of the meaning of a tree and all of its subtrees.

One of the base cases of the induction is when $c = (x := e)$. In this case, we do a case analysis on $x \in \mathbf{d}$. In the case $x \notin \mathbf{d}$, we use a lemma whose conclusion is $?(e = (e \circ \gamma))$, which says that possibly e's value is independent of γ. The lemma is proved by induction on the representability of e. In the case $x \in \mathbf{d}$, we do a case analysis on the representation of e. In the cases where e is a constant or a variable, we know that $R[True, hom_{\gamma}](x := e)$. In the other cases, we prove $?R[True, hom_{\gamma}](x := e)$ the trivial way, by introducing the right disjunct of the definition of ?. The hardest part of the proof of Theorem 2 is the cases for the commands for sending and receiving messages, where we have to make a correspondence between message data references and the argument list of the command.

To obtain a program that computes abstracted specifications, we must show that the pseudo-abstract program $\gamma(P)$ is representable. This involves showing that its initial condition can be expressed as a property on states, and that each of the commands $(\gamma \circ c)$ of $\gamma(P)$ can be represented as a command in the guarded command language. The program representing $\gamma(P)$ need only be equivalent to $\gamma(P)$ on collapsed states, that is, states in the image of the function γ. We express this notion formally via a predicate on states, denoted im_{γ}, defined by $im_{\gamma}(s)$ iff $\exists s' : state.\ s = \gamma(s')$. We need the additional condition on φ that for any expression that is representable, $(\varphi \circ e)$ is also representable.

Theorem 3. *Suppose that ϕ has the additional property that for any expression e, $R[True, True](e)$ implies $R[True, True](\varphi \circ e)$. If $R[True, hom_\gamma](P)$, then $R[im_\gamma, True](\gamma(P))$.*

The proof is by induction on $R[True, hom_\gamma](P)$. For the case $c = (x := e)$, if $x \in \mathbf{d}$, then we use the fact that $(\gamma \circ c) = (\gamma \circ (c \circ \gamma))$ to show that $(\gamma \circ (x := e))$ is equivalent to $x := (\varphi \circ e)$. Because of the assumption on φ, we know that $x := (\varphi \circ e)$ is representable.

8 Conclusion

Using the example of data-value abstraction, we have verified the correctness of an abstraction method for specifications satisfying a particular sufficient condition on their syntax. We have exploited the constructivity of Nuprl to extract a program which can compute the abstract specification corresponding to any concrete specification satisfying the sufficient condition. We were able to do so using only semantic reasoning.

It is unlikely that the approach of dealing with syntax implicitly will always be preferable. This is not a problem, since it can easily coexist with the explicit approach. For example, we could define a conventional recursive type of abstract syntax trees, write a meaning function, and prove that for every representable program there is a tree whose meaning is the program.

Our work was complicated somewhat by our choice of protocol language and its formalization. In particular, since commands are functions on states, instead of relations, non-deterministic commands cannot be represented. With non-determinism, one can include a command that non-deterministically chooses one from an indexed set of commands — this would have been a more natural choice for the actions s_d, and would have obviated the need, in the sufficient condition for symmetry, for finding actions that are similar up to constants on right-hand sides of assignments. Another complication due to the protocol language is its lack of types. This language was developed inside Nuprl to support reasoning about particular protocols, and not for metareasoning about programs.

The precise form of inductive definition mechanism implemented in standard Nuprl [4] is not valid in the classical extension [8] used here. It is not too difficult to adapt it, but we have not done so yet and hence have simply axiomatized the two inductive definitions we needed.

We believe that our results can be extended to deal with temporal properties in the same way as programs. One difficulty is dealing with binding expressions such as universal quantification. It might be possible to deal with universal quantification by using a program variable in place of the quantified variable. We should also be able to extend the results to data-path abstraction, for instance by collapsing the queue in our example.

It should be possible to use our techniques in other theorem provers based on constructive type theory. Classical theorem provers could also formalize the same notion of representability, but it would likely be much less useful, since representability would not encode syntax, and theorems whose conclusion is an application of the possibility operator would be vacuous.

Acknowledgements. The authors would like to thank Bob Kurshan for suggesting the example data-abstraction problem and for useful discussions. The third author would also like to thank Bell Labs for providing the opportunity to work on this problem through a summer internship.

References

1. R. Boulton, A. Gordon, M. Gordon, J. Harrison, J. Herbert, and J. Van Tassel. Experience with embedding hardware description languages in HOL. In *International Conference on Theorem Provers in Circuit Design*, pages 129–156. North-Holland, 1992.
2. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Proc. 19th Ann. ACM Symp. on Principles of Prog. Lang.*, Jan. 1992.
3. R. L. Constable. A note on complexity measures for inductive classes in constructive type theory. *Information and Computation*, 143(2):137–153, 1998.
4. R. L. Constable, et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.
5. J. Dingel and T. Filkorn. Model checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving. In *Seventh International Conference on Computer Aided Verification*, pages 54–69. Springer-Verlag Lecture Notes in Computer Science, 1995.
6. A. P. Felty, D. J. Howe, and F. A. Stomp. Protocol verification in Nuprl. In *Tenth International Conference on Computer Aided Verification*, pages 428–439. Springer-Verlag Lecture Notes in Computer Science, June 1998.
7. K. Havelund and N. Shankar. Experiments in theorem proving and model checking for protocol verification. In *Formal Methods Europe*, pages 662–681. Springer-Verlag Lecture Notes in Computer Science, 1996.
8. D. J. Howe. Semantics foundations for embedding HOL in Nuprl. In *Algebraic Methodology and Software Technology*, pages 85–101. Springer-Verlag Lecture Notes in Computer Science, 1996.
9. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer Verlag, 1991.
10. O. Müller. *A Verification Environment for I/O Automata Based on Formalized Meta-Theory.* PhD thesis, Technische Universität München, 1998.
11. S. Rajan, N. Shankar, and M. K. Srivas. An integration of model checking with automated proof checking. In *Seventh International Conference on Computer Aided Verification*, pages 84–97. Springer-Verlag Lecture Notes in Computer Science, 1995.
12. C. Sprenger. A verified model checker for the modal μ-calculus in Coq. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 167–182. Springer-Verlag Lecture Notes in Computer Science, 1998.
13. P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proc. 13th Ann. ACM Symp. on Principles of Prog. Lang.*, Jan. 1986.

A Formalization of Static Analyses in System $\boldsymbol{F}$

Frédéric Prost

LIP, Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon
Cedex 07 (France)
`Frederic.Prost@ens-lyon.fr`

Abstract. In this paper, we propose a common theoretical framework for type based static analyses. The aim is the study of relationships between typing and program analysis.
We present a variant of Girard's System F called $F^{\Pi}_{\leq:}$. We prove standard properties of $F^{\Pi}_{\leq:}$. We show how it can be used to formalize various program analyses like binding time and dead code. We relate our work to previous analyses in terms of expressivness (often only simply typed calculi are considered) and power (more information can be inferred).
$F^{\Pi}_{\leq:}$ features polymorphism as well as subtyping *at the level* of universe extending previous author work where only universe polymorphism (on a simply typed calculus) was considered

1 Introduction

Static analyses of functional programs such as dead code or binding time are important to perform valuable program optimizations. Many type inference based systems have been proposed to perform those analyses. This work is an attempt to provide a uniform approach for type inference based systems.

1.1 Static Functional Program Analysis

Two approaches are used for static analysis in the functional programming world: semantic based approach, via abstract interpretation (see [CC77,Hun91]), and inference based approach. In this paper, we focus on inference based approach. It has become more and more popular during the early '90s. Many annotated type systems have been proposed (see for instance [Hei95,Sol95,NSN94,HM94,DP98], [TJ92]) for many analyses like Binding Time, Strictness, Dead Code, Control Flow etc. Common features can be found in all these systems. A typed programming language is given. The type system is then modified by the addition of annotations on types or term constructors. Those annotations denote semantical properties of the programs (for a survey see the introduction of [Sol95]). The general picture of this approach is as follows: the programmer writes a typed program, then his program is automatically re-typed under a slightly different type system which includes annotations. In this setting, typing information is meant to be useful for the analysis.

Curry-Howard isomorphism can be used to build programs from proof of their specifications. Nevertheless, code produced this way contain a lot of useless parts from an algorithmic point of view. Indeed, a program just contains what is worthy to implement an algorithm. On the contrary a proof formalizes a great deal of information which is not needed to compute the final result. For instance a proof of the Euclidean division gives for any integers a, b a couple q, r which satisfy $a = bq + r$ and $r \leq b - 1$. From a computational point of view only q, r are valuable, the proof of their properties might be seen as dead code.

The work of C. Paulin (see [Pau89,PM89]) might be seen as a forerunner of type based systems for this kind of dead code analysis. In this work a system to extract F_ω programs from Calculus of Constructions (CC for short) proofs is developed. From a programming language point of view, there is no difference between CC and F_ω. Dependent types of CC can only be used to reason about program properties. Therefore, a syntactical difference between F_ω and CC parts of a term is introduced. It is done by the duplication of the calculus: one part, typed on universe *Prop* is used to denote purely logical parts while the other one typed on universe *Spec* denotes computational parts of the term (parts which may be typed on F_ω). The *extraction* process consists in the erasure of parts typed on *Prop*. It can also be seen as dead code removal.

A major drawback of typed systems lies in their lack of flexibility. For instance consider:

$$p = (\lambda x^{\mathcal{N}}.(+ \ \ (\lambda y^{\mathcal{N}}.5 \ \ x) \ \ x) \ \ 4)$$

One may like to prove that the first occurrence of x is dead, hence typed on *Prop* (if we adopt [PM89] conventions). The problem comes from the second occurrence of x which is alive and hence should be typed on *Spec*. Now, since a term must have a single type and since the analysis must be conservative, x has to be typed on *Spec*. Eventually, in order to keep type consistency, the first occurrence of x cannot be proved dead. A method to overcome this problem, following [Ber96], is to introduce a dummy term $\emptyset$, which is the single inhabitant of a dummy type $\mathcal{U}$ ($\mathcal{N}$ might be seen has the type of naturals built over *Spec* and $\mathcal{U}$ the type of naturals built over *Prop*). The term:

$$p' = (\lambda x^{\mathcal{N}}.(+ \ \ (\lambda y^{\mathcal{U}}.5 \ \ \emptyset) \ \ x) \ \ 4)$$

is well typed. p' can be proved equivalent to p (see [Ber96]), and p' is an improved version of p since it contains less code.

Limitations of type inference based systems come from redexes: $(\lambda f.t \ \ t')$. Variable f must be of the same type than t', but may be used in different situations in t, and each of this situation may require a specific type. Consider for instance:

$$p_1 = (\lambda f^{\mathcal{N} \to \mathcal{N}}.(g \ \ f \ \ (f \ \ 4)) \ \ \lambda x^{\mathcal{N}}.3)$$

where g is variable of type $(\mathcal{N} \to \mathcal{N}) \to (\mathcal{N} \to \mathcal{N})$. Dead code analysis shows that f is a constant function, therefore that 4 is dead. Now, following [Ber96], if we replaced dead code by the dummy constant, we would obtain the optimized program:

$$p_1' = (\lambda f^{\mathcal{U} \to \mathcal{N}}.(g \ \ f \ \ (f \ \ \emptyset)) \ \ \lambda x^{\mathcal{U}}.3)$$

p'_1 is ill typed since f is the argument of g and so should be of type $\mathcal{N} \rightarrow \mathcal{N}$ while f is of type $\mathcal{U} \rightarrow \mathcal{N}$.
To overcome these limitations several paths have been explored. In particular, subtyping (see. [BB95]), ML polymorphism (see [DHM95,Pro97]), conjunctive types (see [DP98]) have been tried out to relax constraints imposed by inference based systems.

1.2 Multi Universe System

We take as programming language Girard's System F (see [GLT89] for an introduction to F), later just called F. This choice is relevant from our theoretical point of view since via impredicative encoding, there is no need to define constants. Naturals, booleans, product types etc. can be built inside the system.

We modify F by the introduction of two different universes from which types may be built. Following C. Paulin, those two universes may be seen as *Prop* and *Spec*. In the following of the paper we will rather use the notation $\perp$ and $\top$ for *Prop* and *Spec*, since our use of those two universes is rather different from the one of [Pau89]. The originality of our system lies on two points:

1. We define an inclusion relation between the two universes, namely $\perp \leq: \top$, from which we derive a subtyping relation on types.
2. We introduce a notion of *universe variable*, which is a refinement of [Pro97] properties variables. We develop a polymorphism à la ML on universes.

It seems sensible that, due to coexistence of subtyping and polymorphism, there may be constraints between universe variables. Consider the following derivation:

$$\cfrac{\cfrac{\Gamma \vdash t : \mathcal{N}^{\alpha_1} \rightarrow \mathcal{N}^{\alpha_2} \qquad \Gamma \vdash x : \mathcal{N}^{\alpha_3} \qquad \mathcal{N}^{\alpha_3} \leq: \mathcal{N}^{\alpha_1}}{\Gamma \vdash (t \;\; x) : \mathcal{N}^{\alpha_2}}}{\vdash \lambda t : \mathcal{N}^{\alpha_1} \rightarrow \mathcal{N}^{\alpha_2}.\lambda x : \mathcal{N}^{\alpha_3}.(t \;\; x) : \mathcal{N}^{\alpha_2}}$$

where $\Gamma \equiv t : \mathcal{N}^{\alpha_1} \rightarrow \mathcal{N}^{\alpha_2}, x : \mathcal{N}^{\alpha_3}$, and $\mathcal{N}^u \equiv \Pi X : u.(X \rightarrow X) \rightarrow X \rightarrow X$. The most general type induced by this derivation, would be likely $\forall \alpha_1, \alpha_2, \alpha_3 .(\mathcal{N}^{\alpha_1} \rightarrow \mathcal{N}^{\alpha_2}) \rightarrow \mathcal{N}^{\alpha_3} \rightarrow \mathcal{N}^{\alpha_2}$. However this scheme would lead to type inconsistency. Indeed if we randomly instantiate this scheme there are no reasons for $\mathcal{N}^{\alpha_3} \leq: \mathcal{N}^{\alpha_1}$ to be satisfied. Therefore, to overcome this problem, we introduce a notion of guarded type scheme: $\forall \alpha.C \Rightarrow (A)$ might be seen as the type A where universe α is abstracted and such that any valid instantiation must satisfy a constraint set C.

1.3 Overview

The type system $F^{\Pi}_{\leq:}$, and its basic properties, are given in section 2. In section 3, we informally discuss the semantics behind $\top$ and $\perp$ universes. Then, in section 4 we see how $F^{\Pi}_{\leq:}$ may be used to formalize program analyses such as dead code and binding time. Finally, in section 5 we relate our work to previous ones and conclude.

2 $F^{\Pi}_{\leq:}$ System

In this section we present a system to reason about type inference based systems for static analyses. It is a multi universe F. The different universes will be used to denote different semantical properties.

2.1 Syntax

We start by defining the syntactic categories of $F^{\Pi}_{\leq:}$:
Universe Variables: Universe variables are α, β elements of an infinite set of universe variables $\mathcal{V}$.

Universes: The set $\mathcal{U}$ of $F^{\Pi}_{\leq:}$ universes is given by the grammar $u ::= \top \mid \bot \mid \alpha$

We define the relation $\leq:$ between universes. We say that $u_1 {\leq:} u_2$ is acceptable unless $u_1 = \top$ and $u_2 = \bot$. In other words, $u_1 {\leq:} u_2$ acceptable only means $u_1 {\leq:} u_2$ is not immediately false.

Type Variables: Type variables noted X, Y, Z are elements of an infinite set of type variable $\mathcal{T}$.

Types: The pre-types of $F^{\Pi}_{\leq:}$ are given by $A, B ::= X \mid A \to B \mid \Pi X : u.A$

Guarded Type Scheme: Guarded Type schemes are given by

$$\sigma ::= A \mid \forall \alpha . C \Rightarrow (\sigma)$$

where C is a set $\{u_0 {\leq:} u_1; \ldots ; u_{2n} {\leq:} u_{2n+1}\}$ of inequalities on $\mathcal{U}$. By extension we say that C is acceptable if for each i in $\{0 \ldots n\}$, $u_{2i} {\leq:} u_{2i+1}$ is acceptable. The reader should be aware that an acceptable constraint set C might have no true closed instance, e.g. $\{\alpha {\leq:} \bot ; \top {\leq:} \alpha\}$. Since quantifiers and constraint sets may only occur at the top level of guarded type schemes, we do not distinguish between $\sigma = \forall \alpha . C \Rightarrow (\forall \beta . C' \Rightarrow (A))$ and $\sigma' = \forall \beta . C' \Rightarrow (\forall \alpha . C \Rightarrow (A))$. Thus, for σ and σ' we write $\forall \alpha, \beta . C \cup C' \Rightarrow (A)$.

Terms: The pre-terms of $F^{\Pi}_{\leq:}$ are given by

$$t, t' ::= x\langle S \rangle \mid (t \;\; t') \mid (t \;\; A) \mid \lambda x : A.t \mid \lambda X : u.t \mid \mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t'$$

where $\mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t'$ might be seen as a synonym for the term $(\lambda x : A.t \;\; t)$, and S is a substitution from universe variables towards $\mathcal{U}$ elements.

Substitutions: $F^{\Pi}_{\leq:}$ substitutions are finite mappings from universe variables to universes:

$$S ::= [\alpha_1 \mapsto u_1, \ldots , \alpha_n \mapsto u_n]$$

where α_i are pairwise distinct. $D(S)$ is the domain of S; it is the set of α_i. $Im(S)$ is the image of S; it is the set of u_i. A substitution S extends naturally to types and terms as follows:

$$[\ldots ; \alpha \mapsto u; \ldots](\alpha) = u$$

$$S(\Pi X : u.A) = \Pi X : S(u).S(A)$$

$$S(X) = X \qquad S(A \to B) = S(A) \to S(B)$$

$$S(x\langle S' \rangle) = x\langle S' \circ S \rangle \qquad S((t\ \ t')) = (S(t)\ \ S(t'))$$

$$S(\lambda x : A.t) = \lambda x : S(A).S(t)\ S(\lambda X : u.t) = \lambda X : S(u).S(t)$$

$$S((t\ \ A)) = (S(t)\ \ S(A))$$

$$S(\mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t') = \mathsf{let}\ x : S(A)\ \mathsf{be}\ S(t)\ \mathsf{in}\ S(t')$$

A substitution S preserves a constraint set $C = \{u_1 \leq: u_2; \ldots; u_n \leq: u_{n+1}\}$ if $S(C) = \{S(u_1) \leq: S(u_2); \ldots; S(u_n) \leq: S(u_{n+1})\}$ is acceptable.

Contexts: A *context* is an ordered sequence given by $\Gamma ::= \emptyset \mid \Gamma, x : A \mid \Gamma, X : u$

Judgments: We define three judgments: $J ::= \Gamma \vdash \ \mid \Gamma \vdash A : u \mid \Gamma \vdash t : A$
The first has to be read "Γ is a well formed context", the second "Under the context Γ, the type A is of universe u" and the third "under the context Γ, the term t is of type A".

Typing Rules: Typing rules are lists of judgment given by $TR ::= \dfrac{J_1 \ldots J_n}{J_{n+1}}$ where $J_1 \ldots J_n$ are premises and J_{n+1} the conclusion.

Derivation Tree: are trees of typing rules. They are given by

$$\Xi ::= TR \mid \frac{\Xi_1 \ldots \Xi_n}{TR}$$

where premises $J_1 \ldots J_n$ of TR are equal to $\Xi_1 \ldots \Xi_n$ conclusions. We write: $\Xi \rhd \Gamma \vdash t : A$, to mean that Ξ is a derivation tree having the conclusion $\Gamma \vdash t : A$.

FV is the set of free type variables and FUV is the set of free universe variables. $Univ(A)$ (resp. $Univ(t)$) is the set of universes occurring in A (resp. t).

Following ML type scheme instantiation and generic instantiation (see [CDDK86] for instance), we define two relations between guarded type schemes.

Definition 1 (Instantiation) *A guarded type scheme σ' is called* an instance of a guarded type scheme σ if there exists a substitution S for free universe variables such that:

$$\sigma' = S(\sigma)$$

Definition 2 (Generic Instantiation) *A guarded type scheme $\sigma' = \forall \beta_1, \ldots, \beta_m.C' \Rightarrow (A'_0)$ is called a* generic instance *of a guarded type scheme $\sigma = \forall \alpha_1, \ldots, \alpha_n.C \Rightarrow (A_0)$ if there exists a substitution S such that:*

1. *$D(S) \subseteq \{\alpha_1, \ldots, \alpha_n\}$,*
2. *$S(C) = C'$ and S preserves C,*
3. *$S(A) = A'$.*

If $\sigma' = A'$ (hence $m = 0$), we write $\sigma \overset{S}{\leadsto} A'$.

$F^{\Pi}_{\leq:}$ is closely related to F. The main difference between F and $F^{\Pi}_{\leq:}$ is that F types are built over a single universe. There is a natural surjection $|.|$ from $F^{\Pi}_{\leq:}$ to F. It is inductively defined as follows:

- **Types:** $|X| = X$, $|A \to A'| = |A| \to |A'|$, and $|\Pi X : u.A| = \Pi X.|A|$.
- **Terms:** $|x\langle S\rangle| = x$, $|(t\ \ t')| = (|t|\ \ |t'|)$, $|(t\ \ A)| = (|t|\ \ |A|)$, $|\lambda x : A.t| = \lambda x : |A|.|t|$, $|\lambda X : u.t| = \lambda X.|t|$, and $|\mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t'| = (\lambda x : |A|.|t|\ \ |t'|)$.

This surjection induces an equivalence relation on $F^{\Pi}_{\leq:}$ types and terms: we say that types A, A' (resp. terms t, t') have the same **skeleton** and write $A \simeq A'$ (resp. $t \simeq t'$), if $|A| = |A'|$ (resp. $|t| = |t'|$).

2.2 Type Inference System

In this section we discuss $F^{\Pi}_{\leq:}$ typing rules. They differ from F typing rules on the following points:

- Types may be built on different universes.
- Based on basic constraints over universes a subtyping relation between types is defined.
- By means of universe variable substitutions, types are related via instantiations and generic instantiations.

We first define the subtyping relation $\leq:$ between types which is an extension of $\leq:$ to types.

Definition 3 (Subtyping) $\leq:$ *between types is defined as follows:*

$$[Ref]\, X \leq: X$$

$$[\to]\ \frac{A_1' \leq: A_1 \qquad A_2 \leq: A_2'}{A_1 \to A_2 \leq: A_1' \to A_2'} \qquad\qquad [\Pi]\ \frac{u_2 \leq: u_1 \qquad A_1[Y := X] \leq: A_2[Z := X]}{\Pi Y : u_1.A_1 \leq: \Pi Z : u_2.A_2}$$

where X is a fresh type variable (it allows $\Pi X : u.X \leq: \Pi Y : u.Y$).

Let $A \leq: A'$, we define $\mathsf{CSet}(A \leq: A')$ as the set of universe constraints used in the proof of $A \leq: A'$. $\mathsf{CSet}(.)$ is defined by induction on the form of A and A':
$\mathsf{CSet}(X \leq: X) = \emptyset$
$\mathsf{CSet}(A_1 \to A_2 \leq: A_1' \to A_2') = \mathsf{CSet}(A_1' \leq: A_1) \cup \mathsf{CSet}(A_2 \leq: A_2')$
$\mathsf{CSet}(\Pi X : u_1.A_1 \leq: \Pi Y : u_2.A_2) = \{u_2 \leq: u_1\} \cup \mathsf{CSet}(A_1 \leq: A_2)$

Well-formed types and well-formed context are mutually inductively defined. Rules are given in Fig. 1.

Well-formed terms are defined by rules given in Fig. 2. A derivation tree Ξ is valid only if it is built with valid judgments.

In Fig. 1 and Fig. 2, C denotes any set of inequalities on $\mathcal{U}$, σ any guarded type scheme, u, u' any universe, A, A' any type, t, t' any term.

– **Contexts:**

$$[Ax]\ \frac{}{\emptyset \vdash}$$

$$[\Gamma T]\ \frac{\Gamma \vdash \quad X \notin FV(\Gamma)}{\Gamma, X : u \vdash} \quad [\Gamma t]\ \frac{\Gamma \vdash \sigma : u \quad x \notin FV(\Gamma)}{\Gamma, x : \sigma \vdash}$$

– **Types:**

$$[\forall C]\ \frac{\Gamma \vdash \sigma : u \quad \alpha \notin FUV(\Gamma)}{\Gamma \vdash \forall \alpha . C \Rightarrow (\sigma) : u}$$

$$[\rightarrow C]\ \frac{\Gamma \vdash A : u \quad \Gamma \vdash A' : u'}{\Gamma \vdash A \rightarrow A' : u'} \qquad [Tvar]\ \frac{\Gamma, X : u \vdash}{\Gamma, X : u \vdash X : u}$$

$$[Tadd]\ \frac{\Gamma \vdash A : u \quad \Gamma, X : u' \vdash}{\Gamma, X : u' \vdash A : u} \quad [tadd]\ \frac{\Gamma \vdash A : u \quad \Gamma, x : A' \vdash}{\Gamma, x : A' \vdash A : u}$$

$$[\Pi C]\ \frac{\Gamma, X : u \vdash A : u'}{\Gamma \vdash \Pi X : u . A : u'}$$

Fig. 1. Rules for well-formed context and types

$$[Hyp]\ \frac{\Gamma, x : \sigma \vdash \quad \sigma \overset{S}{\rightsquigarrow} A}{\Gamma, x : \sigma \vdash x\langle S \rangle : A} \qquad [add]\ \frac{\Gamma \vdash t : A \quad \Gamma, x : \sigma \vdash}{\Gamma, x : \sigma \vdash t : A}$$

$$[\rightarrow I]\ \frac{\Gamma, x : A' \vdash t : A}{\Gamma \vdash \lambda x : A' . t : A' \rightarrow A} \qquad [\Pi I]\ \frac{\Gamma, X : u \vdash t : A}{\Gamma \vdash \lambda X : u . t : \Pi X : u . A}$$

$$[\rightarrow E]\ \frac{\Gamma \vdash t_1 : A_2 \rightarrow A_1 \quad \Gamma \vdash t_2 : A_2' \quad A_2' {\leq:} A_2}{\Gamma \vdash (t_1 \ \ t_2) : A_1}$$

$$[\Pi E]\ \frac{\Gamma \vdash t : \Pi X : u . A \quad \Gamma \vdash A' : u' \quad u' {\leq:} u}{\Gamma \vdash (f \ \ A') : A[X := A']}$$

$$[Poly]\ \frac{\Xi \rhd \Gamma \vdash t : A \quad \Gamma, x : \mathtt{Gen}(\Xi) \vdash t' : A'}{\Gamma \vdash \mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t' : A'}$$

Fig. 2. $F^{\Pi}_{\leq:}$ Typing rules

The intuition behind the function Gen(.), is more or less the same than the one behind ML generalization. The idea is that the most general type of a term is obtained by abstraction of its free variables (here universe variables). Because of subtyping, we cannot simply abstract over free universe variables, since there may be constraints between universe variables that may be not satisfied by random instantiations.

Gen is defined in a mutual inductive way with the term typing judgments: if $\Xi \triangleright \Gamma \vdash t : A$, then

$$\mathtt{Gen}(\Xi) = \begin{cases} \forall \alpha_1, \ldots, \alpha_n . \mathsf{CSet}(\Xi) \Rightarrow (A) & FUV(A) \setminus FUV(\Gamma) = \{\alpha_1, \ldots, \alpha_n\} \\ A \text{ if n=0} \end{cases}$$

where $\mathsf{CSet}(\Xi)$ is the extension of $\mathsf{CSet}(.)$ to typing rules. More precisely:

$$\mathsf{CSet}(\frac{\Xi \triangleright \Gamma, x : \sigma \vdash \quad \sigma \overset{S}{\leadsto} A}{\Gamma, x : \sigma \vdash x\langle S \rangle : A}) = S(C)$$

if $\sigma = \forall \alpha_1, \ldots, \alpha_n . C \Rightarrow (A_0)$.

$$\mathsf{CSet}(\frac{\Xi \triangleright \Gamma \vdash t : A \quad \Gamma, x : \sigma \vdash}{\Gamma, x : \sigma \vdash t : A}) = \mathsf{CSet}(\Xi)$$

$$\mathsf{CSet}(\frac{\Xi \triangleright \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A . t : A \to B}) = \mathsf{CSet}(\Xi)$$

$$\mathsf{CSet}(\Xi = \frac{\Xi \triangleright \Gamma, X : u \vdash t : A}{\Gamma \vdash \lambda X : u . t : \Pi X : u . A}) = \mathsf{CSet}(\Xi)$$

$$\mathsf{CSet}(\frac{\Xi \triangleright \Gamma \vdash t_1 : A_2 \to A_1 \quad \Xi' \triangleright \Gamma \vdash t_2 : A_2' \quad A_2' \leq: A_2}{\Gamma \vdash (t_1 \ \ t_2) : B})$$
$$= \mathsf{CSet}(\Xi) \cup \mathsf{CSet}(\Xi') \cup \mathsf{CSet}(A' \leq: A)$$

$$\mathsf{CSet}(\frac{\Xi \triangleright \Gamma \vdash t : \Pi X : u . A \quad \Xi' \triangleright \Gamma \vdash A' : u' \quad u' \leq: u}{\Gamma \vdash (t \ \ A') : A[X := A']})$$
$$= \mathsf{CSet}(\Xi) \cup \mathsf{CSet}(\Xi') \cup \{u' \leq: u\}$$

$$\mathsf{CSet}(\frac{\Xi \triangleright \Gamma \vdash t : A \quad \Xi' \triangleright \Gamma, x : \mathtt{Gen}(\Xi) \vdash t' : A'}{\Gamma \vdash \mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t' : A'}) = \mathsf{CSet}(\Xi) \cup \mathsf{CSet}(\Xi')$$

The mutually inductive definition of $\mathtt{Gen}(\Xi)$ with term typing rules is sound since derivation trees are finite and recursive calls are done on strictly smaller derivation trees.

We say that a substitution S preserves a derivation tree $\Xi \triangleright \Gamma \vdash t : A$ if S preserves $\mathsf{CSet}(\Xi)$.

The relation between F and $F^{\Pi}_{\leq:}$ is extended to typing derivation. We denote F judgments by $\vdash_F$. F typing rules are simply obtained from $F^{\Pi}_{\leq:}$ rules by erasing all universes. We have the following trivial fact:

Fact 1 *If $\Gamma \vdash t : A$ is a valid derivation tree of $F^{\Pi}_{\leq:}$ then $|\Gamma| \vdash_F |t| : |A|$.*

Conversely, for a given F well formed term, t_F, there exists many well formed $F^{\Pi}_{\leq:}$ terms t', such that $|t'| = t_F$. One can remark that "many" means at least 2: there are the extreme cases when universes used are either all $\top$ or all $\bot$.

2.3 System Properties

Definition 4 (Reductions) *The $F^{\Pi}_{\leq:}$ β-reduction is defined as follows:*

$$(\lambda x : A.t \;\; t') \rightarrow_\beta t[x\langle S\rangle := S(t')] \qquad (\lambda X : b.t \;\; A) \rightarrow_\beta t[X := A]$$

The only noticeable feature of $F^{\Pi}_{\leq:}$ reductions lies in the use of substitutions for let redexes.

We write $=_\beta$ the symetric, transitive closure of $\rightarrow_\beta$, and $\rightarrow^*_\beta$ the reflexive,transitive closure of $\rightarrow_\beta$.

We now study basic properties of $\rightarrow_\beta$ on $F^{\Pi}_{\leq:}$ well formed terms. First we make a straightforward comment regarding relations between $F^{\Pi}_{\leq:}$ reductions and F β-reductions.

Fact 2 *Let t be a well formed $F^{\Pi}_{\leq:}$ term. If $t \rightarrow_\beta t'$ then $|t| \rightarrow_\beta |t'|$ in F.*

From this fact it is easy to prove $F^{\Pi}_{\leq:}$ strong normalization.

Theorem 1 (Strong Normalization) *$\rightarrow_\beta$ is strongly normalizing on $F^{\Pi}_{\leq:}$ terms.*

The Church-Rosser property is not as easy to prove. Indeed, in general substitutions do not commute, hence no simple adaptation of F proof should be expected. However, substitutions occuring in $F^{\Pi}_{\leq:}$ terms are not random ones. It is easy to see from rules of Fig. 2 that domains of substitutions are bound variables. So after a correct renaming of bound variables this problem is solved.

Theorem 2 (Church Rosser) *Let t, t_1, t_2 be well formed $F^{\Pi}_{\leq:}$ terms such that $t \rightarrow_\beta t_1$ and $t \rightarrow_\beta t_2$, then there exist a term t_3 such that $t_1, t_2 \rightarrow^*_\beta t_3$.*

We now study typing derivation properties. We first address the question of typing judgment stability through universe substitutions.

Theorem 3 (Stability of Typing Judgments)

- *Let $\Gamma \vdash A : u$, and S a substitution. Then $S(\Gamma) \vdash S(A) : S(u)$.*
- *Let $\Xi \rhd \Gamma \vdash t : A$ be a valid derivation tree of $F^{\Pi}_{\leq:}$, and S a substitution preserving Ξ, then $S(\Gamma) \vdash S(t) : S(A)$ is a valid $F^{\Pi}_{\leq:}$ judgment.*

It is clear that the subject reduction property does not hold in $F^{\Pi}_{\leq:}$. Indeed, the term $t = (\lambda x : \mathcal{N}^\perp.x \;\; y)$ is well formed of type $\mathcal{N}^\perp$ under a context Γ with $y : \mathcal{N}^\top \in \Gamma$, but $t \rightarrow_\beta y$ and $\Gamma \vdash y : \mathcal{N}^\perp$ is not a valid judgment. Nevertheless, a weaker version of subject reduction holds.

Theorem 4 ("Weak" Subject Reduction) *Let $\Gamma \vdash t : A$ be a valid judgment and $t \rightarrow_\beta t'$, then $\Gamma \vdash t' : A'$ is a valid judgment for some $A' \leq: A$.*

There is also a notion of most general typing in $F^{\Pi}_{\leq:}$:

Theorem 5 (Most General Derivation) *For any well formed F term t under a context Γ, there exists a $F^{\Pi}_{\leq:}$* **most general derivation** *(MGD for short): $\Xi_{gen} \rhd \Gamma_{gen} \vdash t_{gen} : A_{gen}$, such that for any valid derivation $\Xi \rhd \Gamma' \vdash t' : A'$ with $|t'| = t$, there exists a substitution S, preserving Ξ_{gen} such that $S(\Gamma_{gen}) = \Gamma'$ and $S(t_{gen}) = t'$ and $S(A_{gen}) = A'$.*

3 Behind the Scene

In this section, we provide the intuitions that led to the definition of a multi universe typing system. We also informally discuss how $F_{\leq:}^{\Pi}$ can be used to formalize static analyses.

One way of interpreting types is to think of them as partial equivalence relation over an untyped domain D. These interpretations, called PER interpretation, combines a way of interpreting untyped lambda terms as elements of some set D with a way of interpreting types as PER on D (see [Mit90]). The underlying idea is to see the PER interpreting a type A as a typed equality between terms of A. Thus, in standard PER interpretations, the partial equivalence considered are restriction of the equality over the domain of the type. Now, imagine that instead of equality, types are interpreted as restrictions of the trivial relation which relates all elements to all. Under this interpretation it would be impossible to distinguish a term of another one. To formalize this notion, we define positive and negative types.

Definition 5 (Positive and Negative Types) *A type A is called* **positive** *(resp.* **negative***), if $Univ(A) = \{\top\}$ (resp. $Univ(A) = \{\bot\}$).*

We claim that **positive** types are interpreted in a standard way whereas **negative** types are interpreted as trivial PER. This idea was used both for dead code analysis [BB95] and binding time analysis [Hun91], in simply typed calculus.

The reason why $F_{\leq:}^{\Pi}$ may be used for program analyses becomes now clear. Suppose that a term t is a function which takes an argument of a negative type to give back a result of positive type. It means, under the previous semantic, that *whatever* is the input of t, its result will remain unchanged. In other words t is a function constant on its argument.

The originality of our approach lies in the use of universe variables. The idea, already present in [Pro97], is that the sharing of universe variables allows to represent constraints across the term. Thus, the most general type of a term might be seen as a description of the minimal requirements for an analysis to be sound. Once done, the analysis schema can be instantiated to several analyses. In the following section we consider dead code and binding time analyses.

4 Program Analysis

In this section we show how $F_{\leq:}^{\Pi}$ can be used for various static analyses. We only consider closed terms, called programs.

4.1 Observational Equality and Simplified Terms

We define **observational equivalence** on $F_{\leq:}^{\Pi}$ programs. Two programs are observationnaly equivalents if no observation (i.e. any program that takes programs as input and give back a boolean) is able to distinguish them.

We define $\mathcal{B}\mathrm{ool}^u \overset{def}{=} \Pi X : u.X \to X \to X$, the impredicative encoding of boolean type. We also define the two inhabitants of this type:

$$\mathsf{True}^u \overset{def}{=} \lambda X : u.\lambda x_1 : X.\lambda x_2 : X.x_1 \text{ and } \mathsf{False}^u \overset{def}{=} \lambda X : u.\lambda x_1 : X.\lambda x_2 : X.x_2$$

Definition 6 (Observational Equivalence) *Let t_1, t_2 be two programs such that $\vdash t_1 : A$ and $\vdash t_2 : A$, then*

$$t_1 =_{obs} t_2 \iff \textit{for all closed } t \textit{ of type } A \to \mathcal{B}ool^\top \quad (t \ \ t_1) =_\beta (t \ \ t_2)$$

For any negative type A we introduce a dummy constant $\mathtt{d}_A$ of type A. It is easy to check that $F^\Pi_{\leq:}$ extended with those constants shares the same properties than $F^\Pi_{\leq:}$.

We intend to replace terms having negative types with dummy constant of the proper type. Negative type shows that the term is dead. Now, if a term t is dead, all of it subterms are dead too. That is why we only consider terms where dummy constants are *maximal* subterms of negative type. This rules out terms of the form $(\mathtt{d}_{A\to B} \ \ t)$. Indeed, this term has type B and since $A \to B$ is negative, B is also negative, thus we have a term of negative type which is not a dummy term. From now on, we suppose that dummy constants are only used in maximal negative terms.

We now introduce a simplification relation based on dummy constants.

Definition 7 *The simplification relation $\sqsubseteq$ on terms is inductively defined on well formed terms as follows:*

- $\mathtt{d}_A \sqsubseteq t$, *for any well formed term of negative type A.*
- $x\langle S\rangle \sqsubseteq x\langle S\rangle$.
- $\lambda x : A.t_1 \sqsubseteq \lambda y : A.t_2$ *if* $t_1[x := z] \sqsubseteq t_2[y := z]$.
- $\lambda X : u.t_1 \sqsubseteq \lambda Y : u.t_2$ *if* $t_1[X := Z] \sqsubseteq t_2[Y := Z]$.
- $(t_1 \ \ t'_1) \sqsubseteq (t_2 \ \ t'_2)$ *if* $t_1 \sqsubseteq t_2$ *and* $t'_1 \sqsubseteq t'_2$.
- $(t_1 \ \ A) \sqsubseteq (t_2 \ \ A)$ *if* $t_1 \sqsubseteq t_2$.
- $\mathsf{let}\ x : t_1\ \mathsf{be}\ A\ \mathsf{in}\ t'_1 \sqsubseteq \mathsf{let}\ x : t_2\ \mathsf{be}\ A\ \mathsf{in}\ t'_2$, *if* $t_1 \sqsubseteq t_2$ *and* $t'_1 \sqsubseteq t'_2$.

The simplification relation states that a term t is a simplified version of a term t' if t is obtained by the substitution of some negative subterms by a dummy constant of the corresponding type.

We now study the behavior of the simplification relation and observational equality.

Lemma 1 *If t, t' are well formed terms of type $\mathcal{B}ool^\top$ in normal form and $t \sqsubseteq t'$ then $t = t'$.*

We now show some sort of commutation between $\sqsubseteq$ and $\to_\beta$.

Theorem 6 *Let $t_1 \sqsubseteq t_2$ be two well formed programs. If $t_2 \to_\beta t'_2$ then, there exists t'_1 such that $t_1 \ \to^*_\beta \ t'_1$, and $t'_1 \sqsubseteq t'_2$.*

As first application of this theorem, we show that well formed programs related by the simplification relation and having type $\mathcal{B}\text{ool}^{\top}$ have the same normal form.

Theorem 7 *Let $t_1 \sqsubseteq t_2$ be well formed programs of type $\mathcal{B}ool^{\top}$. Then $t_1 =_{\beta} t_2$.*

4.2 Binding Time Analysis

As stated in [Hun91], "given a description of the parameters in a program that will be known at partial evaluation time, a binding time analysis must determine which parts of the program are dependent solely on these parts (and therefore also known at partial evaluation time)". We show that if a function t of positive type, takes a term of negative type A, then the application of this function to *any term* of type A are observationnaly equivalent. In other words: *t is static relatively to its argument.*

Theorem 8 *If t is a program of type $A_1 \to A_2$, with A_1 a negative type and A_2 a positive type. Then, for any t_1, t_2 programs of type A_1, we have $(t\ \ t_1) =_{obs} (t\ \ t_2)$.*

4.3 Dead Code Analysis

Instead of being based on its inputs, as binding time analysis is, Dead code analysis is rather based on its output. Indeed, it aims at finding what part of the program may be removed *without* altering its output. For this analysis, we use [Ber96] principle. In our setting it amounts to show that all negative subterms of a positive program might be seen as dead code.

Theorem 9 *Let $t_1 \sqsubseteq t_2$ be two well formed programs of positive type A. Then, $t_1 =_{obs} t_2$.*

Theorem 9 states that if we replace a negative subterm of a positive program with a dummy constant, the observational behavior of the program does not change. It means that negative subterms are dead since their value has no influence over the observational behavior of the program.

4.4 Examples

Consider the following F program:

$$p_0 \overset{def}{=} \mathsf{let}\ x : \mathcal{N}^{\beta}\ \mathsf{be}\ p\ \mathsf{in}\ (+\ \ x\ \ (\lambda y : \mathcal{N}.5\ \ x))$$

in which $\mathcal{N} \overset{def}{=} \Pi X.(X \to X) \to X \to X$ is the impredicative encoding of natural types, the constant 5 has to be red $\lambda X.\lambda s : X \to X.\lambda z : X.(s\ \ (s \ldots z))$ as the impredicative encoding of this integer, and p a closed term of type $\mathcal{N}$.

In p_0 the variable x is used in 2 places. Its second occurrence is dead since it is the argument of a constant function. Consider the following $F^{\Pi}_{\leq:}$ derivation of this program:

$$\dfrac{\varXi \quad \dfrac{\dfrac{\varGamma \vdash + : \mathcal{N}^{\alpha_0} \rightarrow \mathcal{N}^{\alpha_0} \rightarrow \mathcal{N}^{\alpha_0} \quad \varGamma \vdash x : \mathcal{N}^{\alpha_0}}{\varGamma \vdash (+\ \ x) : \mathcal{N}^{\alpha_0} \rightarrow \mathcal{N}^{\alpha_0}} \quad \dfrac{\dfrac{\varGamma; y : \mathcal{N}^{\alpha_1} \vdash 5 : \mathcal{N}^{\alpha_0}}{\varGamma \vdash \lambda y : \mathcal{N}^{\alpha_1}.5 : \mathcal{N}^{\alpha_1} \rightarrow \mathcal{N}^{\alpha_0}} \quad \dfrac{}{\varGamma \vdash x : \mathcal{N}^{\alpha_1}}}{\varGamma \vdash (\lambda y : \mathcal{N}^{\alpha_1}.5\ \ x) : \mathcal{N}^{\alpha_0}}}{\varGamma \vdash (+\ \ x\ \ (\lambda y : \mathcal{N}^{\alpha_1}.5\ \ x)) : \mathcal{N}^{\alpha_0}}}{\vdash \mathsf{let}\ x : \mathcal{N}^{\beta}\ \mathsf{be}\ p\ \mathsf{in}\ (+\ \ x\ \ (\lambda y : \mathcal{N}^{\alpha_1}.5\ \ x)) : \mathcal{N}^{\alpha_0}}$$

where $\varXi$ denotes a derivation tree for p (we suppose, for the sake of simplicity that $\varXi$ generates no constraints on β), and $\varGamma \equiv x : \forall\beta.\{\} \Rightarrow (\mathcal{N}^{\beta})$. We call this program p_0^*. Now, consider the following substitution: $S = [\alpha_0 \mapsto \top; \alpha_1 \mapsto \bot]$.

If we apply this substitution to the derivation tree of p_0^*, we obtain:

$$\dfrac{S(\varXi) \quad \dfrac{\dfrac{\varGamma \vdash + : \mathcal{N}^{\top} \rightarrow \mathcal{N}^{\top} \rightarrow \mathcal{N}^{\top} \quad \varGamma \vdash x : \mathcal{N}^{\top}}{\varGamma \vdash (+\ \ x) : \mathcal{N}^{\top} \rightarrow \mathcal{N}^{\top}} \quad \dfrac{\dfrac{\varGamma; y : \bot \vdash 5 : \mathcal{N}^{\top}}{\varGamma \vdash \lambda y : \mathcal{N}^{\bot}.5 : \mathcal{N}^{\bot} \rightarrow \mathcal{N}^{\top}} \quad \dfrac{}{\varGamma \vdash x : \mathcal{N}^{\bot}}}{\varGamma \vdash (\lambda y : \mathcal{N}^{\bot}.5\ \ x) : \mathcal{N}^{\top}}}{\varGamma \vdash (+\ \ x\ \ (\lambda y : \mathcal{N}^{\bot}.5\ \ x)) : \mathcal{N}^{\top}}}{\vdash \mathsf{let}\ x : \mathcal{N}^{\beta}\ \mathsf{be}\ p\ \mathsf{in}\ (+\ \ x\ \ (\lambda y : \mathcal{N}^{\bot}.5\ \ x)) : \mathcal{N}^{\top}}$$

Now, since $S(p_0^*)$ is of positive type, and since the second occurrence of x in $S(p_0^*)$ is of negative type theorem 9 states that it may safely be replaced with a dummy term. Let $p_{simpl} = \mathsf{let}\ x : \mathcal{N}^{\beta}\ \mathsf{be}\ p\ \mathsf{in}\ (+\ \ x\ \ (\lambda y : \mathcal{N}^{\bot}.5\ \ \mathtt{d}_{\mathcal{N}^{\bot}}))$, then $p_{simpl} \sqsubseteq p_0^*$ and is of type $\mathcal{N}^{\top}$. Therefore by theorem 9 we have proved the second occurrence of x to be dead.

This trivial example illustrates well the mechanisms of program analyzes in $F^{\Pi}_{\leq:}$. We left as exercise to the reader the following program:

$$\begin{array}{rl} p_1 \overset{def}{=} & (\lambda F^{(\mathcal{N}\rightarrow\mathcal{N}\rightarrow\mathcal{N})\rightarrow\mathcal{N}\rightarrow\mathcal{N}\rightarrow\mathcal{N}}.(+\ \ (F\ \ \lambda x^{\mathcal{N}}, y^{\mathcal{N}}.x\ \ m\ \ n) \\ & \qquad\qquad\qquad\qquad\qquad\qquad (F\ \ \lambda x^{\mathcal{N}}, y^{\mathcal{N}}.y\ \ o\ \ q)) \\ & \lambda f^{\mathcal{N}\rightarrow\mathcal{N}\rightarrow\mathcal{N}}, z^{\mathcal{N}}, v^{\mathcal{N}}.(f\ \ z\ \ v)) \end{array}$$

where m, n, o, q are any closed term of type $\mathcal{N}$. Types are given in superscript for notational convenience. Using $F^{\Pi}_{\leq:}$ it is possible to prove that n, o are dead code. It is impossible to prove it using [BB95].

Another example worth of consideration is the following one:

$$\begin{array}{rl} p_2 \overset{def}{=} & (\lambda F^{(\mathcal{N}\rightarrow\mathcal{N})\rightarrow\mathcal{N}\rightarrow\mathcal{N}}.(F\ \ \lambda y^{\mathcal{N}}.5\ \ m) \\ & \lambda f^{\mathcal{N}\rightarrow\mathcal{N}}, x^{\mathcal{N}}.(f\ \ (f\ \ x))) \end{array}$$

where m is any closed term of type $\mathcal{N}$. Using $F^{\Pi}_{\leq:}$ it is possible to prove that m is dead code while it is impossible to prove it in [Pro97].

5 Conclusion

We have presented a variant of system F in which types are build on different universes. ML-Polymorphism at the level of universe as well as subtyping, derived from a basic inclusion between $\top$ and $\bot$, are available in $F^{\Pi}_{\leq:}$.We have shown the basic properties of this system, exposed how it can be used to formalize program analyzes (construction of the most generic typing), and studied applications for dead code and binding time.

Our work differs from earlier studies where polymorphism and/or subtyping are used for program analyzes. First, our underlying type system is a polymorphic λ-calculus, whereas previous works only consider simply typed lambda-calculi (except [Boe94] but there is no polymorphism nor subtyping in it). It makes possible to avoid *ad hoc* typing rules for constants such as if then else construct, natural numbers, booleans and basic operations on them (like Successor,+ etc), through their impredicative encoding. Second, previous works are generally linked (excepted [Ber96,BB95,Boe94]) to a given reduction strategy. In our setting only β-reduction is considered. It makes our study valid under any reduction strategy.

From a theoretical point of view, this work provides an abstract approach of type based program analyses. It enlightens the relationships between type inference and program analyses. By few means (different universes and an inclusion rule between universes), we have shown that it is possible to formalize different analyses.

Possible further work includes the definition of a suitable semantics for $F^{\Pi}_{\leq:}$, based on the intuition developed in section 3. There are also no fundamental reasons to limit the number of universes to two. For strictness and totality analysis for instance one could imagine a variant of $F^{\Pi}_{\leq:}$ with three universes. Another direction is to try the idea of variable universes to F_ω or even the Calculus of Constructions. Indeed, the Calculus of Construction is the underlying type system of proof checkers as Coq (see [BBC+96]) and Lego (see [LP92]). These proof checkers allow to build programs as proof of their specifications through *extraction*. Extraction might more or less be seen as dead code elimination (which is not entirely true since extraction provides a proof that the extracted program verifies its specification). It is likely that our dead code analysis would improve existing extractions.

Acknowledgments

I thank Christine Paulin who suggested the inclusion $\bot\leq:\top$, I also thank Stefano Berardi for discussions where he gave me valuable intuitions about the implications of the inclusion for dead code analysis.

References

[BB95] S. Berardi and L. Boerio. Using subtyping in program optimization. In *Proceedings of TLCA'95, LNCS 902*. Spinger-Verlag, 1995.

[BBC$^+$96] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliâtre, H. Herbelin, G. Huet, P. Manoury, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi, and B. Werner. *The Coq Proof Assistant Reference Manual Version 6.1.* INRIA-Rocquencourt-CNRS-ENS Lyon, December 1996.

[Ber96] S. Berardi. Pruning simply typed lambda terms. *Journal of Logic and Computation*, 125(2):663–681, 15 March 1996.

[Boe94] L. Boerio. Extending pruning techniques to polymorphic second order λ–calculus. In D. Sanella, editor, *Proceedings of ESOP'94, LNCS 788*, pages 120–134. Springer-Verlag, April 1994.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages* (*POPL '77*), pages 238–252, New York, 1977. ACM Press.

[CDDK86] D. Clément, J. Despeyroux, T. Desperoux, and G. Kahn. A simple applicative language: Mini-ML. Technical Report 529, INRIA-Sophia Antipolis, May 1986.

[DHM95] D. Dussart, F. Henglein, and C. Mossin. Polymorphic recursion and subtype qualifications: Polymorphic binding-time analysis in polynomial time. In Alan Mycroft, editor, *SAS'95: 2nd Int'l Static Analysis Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 118–135, Glasgow, Scotland, September 1995. Springer-Verlag.

[DP98] F. Damiani and F. Prost. Detecting and removing dead code using rank-2 intersection. In *International Workshop:"TYPES'96", selected papers, LNCS 1512*. Spinger-Verlag, 1998.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[Hei95] N. Heintze. Control-flow analysis and type systems. In Alan Mycroft, editor, *Proceeding of SAS 1995, LNCS 983*, pages 189–206. Springer-Verlag, 1995.

[HM94] C. Hankin and D. Le Métayer. A type-based framework for program analysis. In *Proceedings of the Static Analysis Symposium, LNCS 864*, pages 380–394. Springer-Verlag, 1994.

[Hun91] S. Hunt. *Abstract Interpretation of functionnal languages : from theory to Practice.* PhD thesis, Department of Computing, Imperial College, London, 1991.

[LP92] Z. Luo and R. Pollack. Lego proof development system : User's manual. Technical Report ECS-LFCS-92-211, University of Edinburgh., 1992.

[Mit90] J.C. Mitchell. A type inference approach to reduction properties and semantics of polymorphic expressions. In G. Huet, editor, *Logical Foundations of Functionnal programming*, pages 195–211. Addison-Wesley, 1990. (Chapter 9).

[NSN94] H.R. Nielson, K.L. Solberg, and F. Nielson. Strictness and totality analysis. In *Static Analysis, LNCS 864*, pages 408–422. Springer-Verlag, 1994.

[Pau89] C. Paulin. *Extraction de programmes dans le calcul des constructions.* PhD thesis, Université Paris 7, January 1989.

[PM89] C. Paulin-Mohring. Extracting F_ω's programs from proofs in the Calculus of Constructions. In *Sixteenth Annual ACM Symposium on Principles of Programming Languages*, Austin, January 1989. ACM.

[Pro97] F. Prost. Using ML type inference for dead code analysis. Research Report RR97-09, LIP, ENS Lyon, France, May 1997.

[Sol95] K. Lackner Solberg. *Annotated Type Systems for Program Analysis.* PhD thesis, Odense University, July 1995.

[TJ92] J.-P. Talpin and P. Jouvelot. The type and effect discipline. In IEEE Computer Society Press, editor, *Proceedings of the 1992 Conference on Logic in Computer Science*, 1992.

On Explicit Reflection in Theorem Proving and Formal Verification

Sergei N. Artemov[⋆]

Department of Computer Science
Cornell University
Ithaca, NY 14853, U.S.A.
artemov@cs.cornell.edu
http://www.cs.cornell.edu/Info/People/artemov

Abstract. We show that the stability requirement for a verification system yields the necessity of some sort of a reflection mechanism. However, the traditional reflection rule based on the Gödel implicit provability predicate leads to a "reflection tower" of theories which cannot be formally verified. We found natural lower and upper bounds on a metatheory capable of establishing stability of a given verification system. The paper also introduces an explicit reflection mechanism which can be verified internally. This circumvents the reflection tower and provides a theoretical justification for the verification process. On the practical side, the paper gives specific recommendations concerning verification of inference rules and building a verifiable reflection mechanism for a theorem proving system.

1 Introduction

There is a large variety of theorem provers and proof checkers which can be used for verification (cf. [8], [1], [11]). The mathematical counterparts of those systems range from first order logic (e.g. in **FOL**) and certain fragments of first order arithmetic to higher order logic (**HOL**), the systems with powerful principles sufficient to accommodate most of the classical mathematics (**Mizar**) and most of the computational and constructive tools (**Nuprl**). The underlying logic of such systems can be either classical or intuitionistic. In this paper we assume that

The degree of confidence in facts verified by a certain system is not higher than the degree of confidence in the system itself.

This paradigm yields the necessity to keep an account of the tools used in a given verification process, including the verification system $\mathcal{V}$ itself along with

[⋆] The research described in this paper was supported in part by ARO under the MURI program "Integrated Approach to Intelligent Systems", grant DAAH04-96-1-0341, by DARPA under program LPE, project 34145.

an exact description of the set of all metamathematical assumptions $\mathcal{M}$ made in the process of verification. Therefore, the set of beliefs which the verification is based upon should include $\mathcal{V} \cup \mathcal{M}$. Without loss of generality we assume in this paper that a metatheory $\mathcal{M}$ of a given verification system $\mathcal{V}$ contains $\mathcal{V}$, therefore, $\mathcal{V} \cup \mathcal{M} = \mathcal{M}$.

For example, suppose we want to verify a statement F by means of the first order arithmetic $\mathcal{PA}$ (i.e. $\mathcal{V} = \mathcal{PA}$). One of the possible ways to put this problem on a formal setting is to say that our goal consists in establishing that $\mathcal{PA} \vdash \mathit{Provable}(F)$, where $\mathit{Provable}(F)$ is a formal statement saying that "F is provable by certain formal tools". Suppose that we have established that $\mathcal{ZF} \vdash \mathit{Provable}(F)$, where $\mathcal{ZF}$ is the Zermelo-Frenkel set theory (a much stronger theory than $\mathcal{PA}$). This corresponds to a realistic situation when a verifier uses the power of all of mathematics, not only the elementary methods formalizable in $\mathcal{PA}$. Here is the sketch of the standard metamathematical argument which under certain assumptions about $\mathcal{ZF}$ concludes that in fact $\mathcal{PA} \vdash \mathit{Provable}(F)$: assume that $\mathcal{ZF}$ is ω-consistent (cf. [15],[7],[16]); since $\mathit{Provable}(F)$ is an arithmetical Σ_1 statement, this yields that $\mathit{Provable}(F)$ is true and, by the Σ_1-completeness of $\mathcal{PA}$, $\mathcal{PA} \vdash \mathit{Provable}(F)$. On the one hand, we have succeeded in establishing that $\mathcal{PA} \vdash \mathit{Provable}(F)$. On the other hand, at the metalevel of this argument we have used the power of $\mathcal{ZF}$ and even the assumption of ω-consistency of $\mathcal{ZF}$. A total account of the beliefs involved in this verification process should include this assumption, which, by the way, has never been and could not possibly be proven by any usual consistent mathematical means.

In this paper we will try to demonstrate the following three points:

1. Some form of the reflection rule is a necessary part of an extendable and stable verification system. This will emerge as a natural corollary of the soundness, extensibility, and stability assumptions (cf. [8]) about a verification system. Moreover, even the most basic proof checking scheme when $\mathcal{V}$ verifies a proof of F and then concludes that F itself holds requires reflection.

2. The traditional reflection based on the implicit provability predicate does not provide a satisfactory justification of formal verification. It is well-known that the implicit reflection in a given system $\mathcal{V}$ cannot be verified in $\mathcal{V}$ (cf. [8], [12], [1], [11]). In particular, this means that implicit reflection cannot justify even the basic proof checking by means of $\mathcal{V}$ without imposing additional unverifiable assumptions on $\mathcal{V}$. The present paper demonstrates that an iterative use of reflection leads to the "reflection tower" of theories which is not computably enumerable and cannot be itself verified by any formal tools. If one takes into account these hidden metamathematical costs of implicit reflection, then no verifiable stable systems exist.

3. There is a verifiable reflection mechanism: "explicit reflection" (introduced in the present paper), which provides a foundational justification of the verification process. Explicit reflection requires more information in order to certify the premises of the reflection rule. However, this additional information is usually available in real processes of verification; the old implicit provability model just has not had tools of its utilization. The explicit reflection circumvents the reflection tower and provides the strict justification of verification. On the practical side, the paper gives specific recommendations concerning the verification of the admissible rules and building a verifiable reflection mechanism for a theorem proving system.

2 Verification Systems

Definition 1. *Under a* **verification system** $\mathcal{V}$ *we will understand a formal theory satisfying the following conditions a) – d):*

a) The underlying logic of $\mathcal{V}$ *is either classical or intuitionistic.*

b) Proofhood in $\mathcal{V}$ *is decidable, therefore theoremhood in* $\mathcal{V}$ *is computably enumerable. Note that by the well-known Craig theorem the former follows from the latter for an appropriate choice of axiom system.*

c) $\mathcal{V}$ *is strong enough to represent any computable function and decidable relation.*

d) $\mathcal{V}$ *has some sort of a numeration of syntax mechanism in the style of [8], [1]. In particular, there is an injective function rep which maps syntactic objects like terms, formulas, finite sequences of formulas, sequents, finite trees labeled by sequents, derivation trees, etc., into standard ground terms of* $\mathcal{V}$*. The usual notation used in this case is* $\ulcorner s \urcorner = rep(s)$*. The function rep and its inverse are both computable. We assume that* $\mathcal{V}$ *is able to derive formalizations of "usual" combinatory properties of the syntactic objects at a level corresponding to the first order intuitionistic arithmetic* $\mathcal{HA}$*.*

For the sake of notational simplicity we will use the same names for the informal objects (relations, functions, numbers) and for their formal counterparts (formulas, terms, ground terms) whenever unambiguous.

Examples of verification systems: the first order arithmetic $\mathcal{PA}$; the first order intuitionistic arithmetic $\mathcal{HA}$ and its extensions; second order arithmetic; Martin-Löf type theory $\mathcal{ITT}$; formal set theory $\mathcal{ZF}$; etc. Note that all the above conditions on $\mathcal{V}$ have a purely constructive syntactic character. We have assumed neither semantic properties of $\mathcal{V}$ (e.g. soundness with respect to some semantics), nor metamathematical ones (consistency, ω-consistency, etc.).

Definition 2. *For any verification system* $\mathcal{V}$ *there is a provably* Δ_1 *formula* $Proof(x,y)$ *in the language of* $\mathcal{V}$ *(called* **a proof predicate***) which is obtained by a natural formalization of the inductive definition of derivation in* $\mathcal{V}$ *(cf. [9], [8], [1]). In particular,* $Proof(\ulcorner \mathcal{D} \urcorner, \ulcorner \varphi \urcorner)$ *holds iff* $\mathcal{D}$ *is a proof of* φ *in* $\mathcal{V}$*. The Gödel* **provability predicate** $Provable(y)$ *is defined as* $\exists x Proof(x,y)$*. We will use the notation* $\Box\varphi$ *for* $Provable(\ulcorner \varphi \urcorner)$ *and* $[\![p]\!]\varphi$ *for* $Proof(p, \ulcorner \varphi \urcorner)$*. For any finite set of* $\mathcal{V}$*-formulas* Γ *by* $\Box\Gamma$ *we mean the conjunction of* $\Box\psi$*'s for all* $\psi \in \Gamma$*.*

Definition 3. *The consistency formula* $Consis(\mathcal{V})$ *is defined as* $\neg\Box\bot$, *where* $\bot$ *is the standard boolean constant FALSE in* $\mathcal{V}$.

The informal meaning of $Consis(\mathcal{V})$ is that there is no a proof of *FALSE* in $\mathcal{V}$: this is one of the equivalent formulations of the consistency assertion of $\mathcal{V}$ in the language of $\mathcal{V}$.

We will refer to the provability predicate $\Box(\cdot)$ as the *implicit provability predicate.* The reason for choosing this name lies in the fact that in the formula $\Box\varphi$ (i.e. $\exists x Proof(x, \ulcorner\varphi\urcorner)$) the proof is represented implicitly by the existential quantifier, which does not provide any specification of this proof.

The implicit provability predicate has been studied extensively since its invention by Gödel in 1930. The milestone results here are the second Gödel incompleteness theorem (cf. [15], [7])), which states that

$$\text{If } \mathcal{V} \text{ is consistent then } \mathcal{V} \nvdash Consis(\mathcal{V}),$$

and the Löb theorem which says that

$$\mathcal{V} \vdash \Box\varphi \rightarrow \varphi \text{ implies } \mathcal{V} \vdash \varphi.$$

By the well-known Hilbert-Bernays lemma (cf. [15],[7]),

$$\mathcal{V} \vdash \varphi \quad \text{implies} \quad \mathcal{V} \vdash \Box\varphi.$$

This lemma can be considered as a justification of the *formalization rule* $\varphi/\Box\varphi$ for $\mathcal{V}$, which states that every proof in $\mathcal{V}$ can be formalized in $\mathcal{V}$. The proof of the formalization rule is purely syntactic and does not involve any extra assumptions about $\mathcal{V}$. Moreover, this rule can be formalized and proven inside $\mathcal{V}$ (cf. [15], [7]):

$$\mathcal{V} \vdash \Box\varphi \rightarrow \Box\Box\varphi.$$

Below we will use one more fact about the provability operator $\Box$, usually attributed to Hilbert, Bernays and Löb (cf. [15],[7]):

$$\mathcal{V} \vdash \Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi).$$

3 Stability Equals Admissibility of Reflection Rule

The basic properties required from a verification system are soundness, extensibility, and stability ([8]). We will discuss soundness in Section 5. Extensibility and stability will appear in this section below.

Definition 4. *A* **rule of inference** R *in the language of* $\mathcal{V}$ *is a computable partial function from a decidable set of lists of* $\mathcal{V}$*-formulas to the set of* $\mathcal{V}$*-formulas.*

The usual notation for a rule of inference R is Γ/φ, where Γ indicates the argument of R (premises), and φ the value $R(\Gamma)$ of R (conclusion). For the sake of notational convenience we will not distinguish between a finite list of formulas Γ and one formula which is the conjunction of all formulas from Γ when unambiguous. We would like to think that such an abuse of notation will be tolerated by a reader.

Definition 5. *A rule of inference Γ/φ in $\mathcal{V}$ is* **derivable** *if for any instance of Γ/φ there is a deduction in $\mathcal{V}$ of its conclusion from its premises.*

A rule of inference Γ/φ in $\mathcal{V}$ is **provable** *if $\mathcal{V} \vdash \Gamma \rightarrow \varphi$.*

A rule of inference Γ/φ in $\mathcal{V}$ is **admissible** *if $\mathcal{V} \vdash \Gamma$ implies $\mathcal{V} \vdash \varphi$.*

Derivable rules is a system with the deduction theorem are provable. It is also natural to assume that every provable rule is derivable since there is a standard deduction of φ from Γ (free premises) and $\Gamma \rightarrow \varphi$ (which has a derivation without premises in $\mathcal{V}$.). To a certain extend the notion of a provable rule is a system-independent version of the notion of a derivable rule.

It is immediate from the definitions that every derivable and every provable rule of inference is admissible. None of the converse implications holds in a general case. It is important to notice that there are serious reasons to assume that derivable and provable rules alone are not sufficient for an advanced system of automated deduction and verification (cf. also [14]).

Example 1. Here are examples of widely used admissible rules that are not provable: $\varphi/\forall x\varphi$ (*generalization*), $\varphi(x)/\varphi(y)$ (*renaming of free variables*), $\neg\neg\sigma/\sigma$ where σ is a Σ_1 sentence (*Markov rule* for intuitionistic arithmetic $\mathcal{HA}$, cf. [17]), *disjunctive* and *existential* rules for intuitionistic systems, $\varphi/\Box\varphi$ (*formalization*), $\Box\varphi \rightarrow \varphi/\varphi$ (*Löb rule*), $\Box\varphi/\varphi$ (*implicit reflection*, for classical and intuitionistic arithmetic), etc.

Extending $\mathcal{V}$ by adding rules provable in $\mathcal{V}$ does not change the system. In turn, stability of $\mathcal{V}$ with respect to admissible rules rises to a serious theoretical and practical problem since some admissible rules cannot be verified inside $\mathcal{V}$.

Definition 6. *A rule Γ/φ in $\mathcal{V}$ is* **verifiable in** $\mathcal{M}$ *if $\mathcal{M} \vdash \Box\Gamma \rightarrow \Box\varphi$. A rule in $\mathcal{V}$ is* **internally verifiable** *if it is verifiable in $\mathcal{V}$.*

Every admissible rule in $\mathcal{V}$ is verifiable in some sufficiently large theory $\mathcal{M}$, but not every one of them is verifiable in $\mathcal{V}$.

Example 2. The rules *generalization, renaming of free variables, formalization, Löb rule, Markov rule* and internally verifiable. The rules *implicit reflection, disjunctive rule, existential rule* are not internally verifiable.

We accept the understanding of stability as conservativity of extensions by internally verifiable rules (cf. [1], [8], [11], [14]).

Definition 7. *System $\mathcal{V}' \supseteq \mathcal{V}$ is* **conservative** *over $\mathcal{V}$ if for any formula ψ*

$$\mathcal{V}' \vdash \psi \quad \textit{implies} \quad \mathcal{V} \vdash \psi.$$

A system $\mathcal{V}$ is **stable** *if for any rule Γ/φ verifiable in $\mathcal{V}$ the system $\mathcal{V} + \Gamma/\varphi$ is conservative over $\mathcal{V}$.*

Definition 8. *By $IRR(\mathcal{V})$ we denote the implicit reflection rule $\Box\varphi/\varphi$ where $\Box\varphi$ represents the provability of φ in $\mathcal{V}$.*

Example 3. Here is the standard example of a formal theory for which the implicit reflection rule is not admissible ([9]): $\mathcal{V} = \mathcal{PA} + \neg Consis(\mathcal{PA})$. This system is consistent, i.e. $\mathcal{V} \nvdash \bot$. On the other hand $\mathcal{V} \vdash \Box\bot$, where $\Box$ stands for provability in this particular $\mathcal{V}$.

Theorem 1. *A verification system $\mathcal{V}$ is stable iff the implicit reflection rule $IRR(\mathcal{V})$ is admissible in $\mathcal{V}$.*

Proof. Let $\mathcal{V}$ be a stable system. Let us consider the rule R_φ consisting of a single pair $(TRUE, \varphi)$, where $TRUE$ is the propositional constant for true statements in $\mathcal{V}$. Since $\mathcal{V} \vdash TRUE$, we also have $\mathcal{V} \vdash \Box(TRUE)$. By stability of $\mathcal{V}$, for all φ, ψ if $\mathcal{V} \vdash \Box(TRUE) \to \Box\varphi$ and $\mathcal{V} + TRUE/\varphi \vdash \psi$, then $\mathcal{V} \vdash \psi$. Equivalently, for all φ, ψ if $\mathcal{V} \vdash \Box\varphi$ and $\mathcal{V} + \varphi \vdash \psi$, then $\mathcal{V} \vdash \psi$. Let ψ be φ. Then $\mathcal{V} \vdash \Box\varphi$ implies $\mathcal{V} \vdash \psi$ for all φ, therefore $IRR(\mathcal{V})$ is admissible in $\mathcal{V}$.

Let now $IRR(\mathcal{V})$ be admissible in $\mathcal{V}$, i.e. $\mathcal{V} \vdash \Box\varphi$ implies $\mathcal{V} \vdash \varphi$, and let Γ/φ be a verified rule, i.e. $\mathcal{V} \vdash \Box\Gamma \to \Box\varphi$. By an induction on the derivation in $\mathcal{V} + \Gamma/\varphi$ we prove that $\mathcal{V} + \Gamma/\varphi \vdash \psi$ implies $\mathcal{V} \vdash \psi$. The induction basis holds because $\mathcal{V}$ and $\mathcal{V} + \Gamma/\varphi$ have the same set of axioms. The induction step in the case of a rule other than Γ/φ is trivial. Let ψ be obtained in $\mathcal{V} + \Gamma/\varphi$ by the rule Γ/φ, i.e. there is specific Γ_1 such that Γ_1/ψ is a special case of the rule Γ/φ and $\mathcal{V} + \Gamma/\varphi \vdash \Gamma_1$. By the induction hypothesis, $\mathcal{V} \vdash \Gamma_1$. By the formalization rule in $\mathcal{V}$, $\mathcal{V} \vdash \Box\Gamma_1$. Since the rule Γ/φ is verified, we have $\mathcal{V} \vdash \Box\Gamma_1 \to \Box\psi$, therefore $\mathcal{V} \vdash \Box\psi$. By the rule $IRR(\mathcal{V})$, $\mathcal{V} \vdash \psi$. ∎

Note 1. Such systems as **LCF, Nuprl, HOL** have the extension mechanisms of *tactics* based on the representation of provable rules. This use of tactics can be justified inside the system and the stability with respect to tactics can be established without any additional assumptions. A general case of stability with respect to all verified rules (including the ones that are not provable) was considered in [1], [8], [11], [14]. It follows from Theorem 1 that the general stability of a verification system $\mathcal{V}$ is equivalent to admissibility of the implicit reflection rule in $\mathcal{V}$.

4 Provability Tests and Other Sources of Reflection

Stability of a verification system with respect to verifiable rules is not the only place where the reflection rule becomes necessary.

The basic proof checking scheme when $\mathcal{V}$ verifies a proof of φ in $\mathcal{V}_1$ and then concludes that $\mathcal{V}_1 \vdash \varphi$ requires some sort of reflection too. In terms of the implicit provability this proof checking scheme is the rule

$$\mathcal{V} \vdash \Box_1\varphi \;\Rightarrow\; \mathcal{V}_1 \vdash \varphi,$$

where $\Box_1$ stands for the provability predicate in $\mathcal{V}_1$. In particular, when $\mathcal{V}_1$ is $\mathcal{V}$ itself this rule transforms into the usual implicit reflection rule $IRR(\mathcal{V})$ for $\mathcal{V}$.

A better internally verifiable theoretical description of the verification scheme above is given in terms of the explicit reflection in Section 7.

Another class of examples that need reflection has been shown to me by Robert Constable and Stuart Allen. These examples are provided by completeness theorems and other provability tests which play important role in theoretical logic and mathematics and which are now within a scope of interests of advanced automated deduction systems.

Definition 9. *A* **provability test** *for $\mathcal{V}$ is a formula TEST(x) such that $\mathcal{V} \vdash$ TEST($\ulcorner\varphi\urcorner$) $\rightarrow \Box\varphi$ for any formula φ. $\mathcal{V}$ is* **stable with respect to provability tests** *if for every provability test TEST(x) and every formula φ*

$$\mathcal{V} \vdash \mathit{TEST}(\ulcorner\varphi\urcorner) \text{ yields } \mathcal{V} \vdash \varphi.$$

In other words, if φ passes a provability test then φ is provable in $\mathcal{V}$.

Example 4. Formalized completeness and decidability theorems may be regarded as provability tests. Indeed, a completeness theorem generally states that if φ is valid with respect to a certain semantics then φ is provable. In its formalized version such a theorem is a formula *VALID*($\ulcorner\varphi\urcorner$)$\rightarrow \Box\varphi$ provable in $\mathcal{V}$, and one can take *VALID(x)* as *TEST(x)*. A formalized decidability theorem usually has the form $\mathcal{V} \vdash$ *TEST*($\ulcorner\varphi\urcorner$) $\leftrightarrow \Box\varphi$, where *TEST(x)* is the formula describing a decidability algorithm working on the code of φ and deciding whether φ is provable.

Theorem 2. *$\mathcal{V}$ is stable with respect to provability tests iff the implicit reflection rule IRR($\mathcal{V}$) is admissible in $\mathcal{V}$.*

Proof. Let *IRR*($\mathcal{V}$) be admissible in $\mathcal{V}$, and *TEST(x)* be a provability test. If $\mathcal{V} \vdash$ *TEST*($\ulcorner\varphi\urcorner$) then $\mathcal{V} \vdash \Box\varphi$ and, by the reflection rule, $\mathcal{V} \vdash \varphi$. Therefore $\mathcal{V}$ is stable with respect to provability tests.

Let now $\mathcal{V}$ be stable with respect to all provability tests. In particular, the stability with respect to the trivial test when *TEST(x)* is *Provable(x)* means that $\mathcal{V} \vdash \Box\varphi$ yields $\mathcal{V} \vdash \varphi$ for every φ, i.e. *IRR*($\mathcal{V}$) is admissible in $\mathcal{V}$.

5 Metamathematical Cost of Soundness and Stability

In this section we will find lower and upper bounds for the minimal metatheory $\mathcal{M}$ capable of establishing soundness and stability of a verification system $\mathcal{V}$.

We will use the Turing progression as the standard scale to measure the metamathematical strength of a given extension of the basic theory ([13]). The Turing progression $\mathcal{V}^c_\alpha$ of theories (cf. [18], [10], [2]) for $\mathcal{V}$ is obtained from $\mathcal{V}$ by iterating the consistency assumptions along the Church-Kleene system of constructive ordinals α.

We consider the first ω theories from the Turing progression.

$$\mathcal{V}_0^c = \mathcal{V}, \qquad \mathcal{V}_{n+1}^c = \mathcal{V}_n^c + Consis(\mathcal{V}_n^c), \qquad \mathcal{V}_\omega^c = \bigcup_n \mathcal{V}_n^c.$$

If $\mathcal{V}$ is correct with respect to the standard model of arithmetic, then the following strict inclusions hold:

$$\mathcal{V}_0^c \subset \mathcal{V}_1^c \subset \mathcal{V}_2^c \subset \ldots \subset \mathcal{V}_\omega^c.$$

Soundness was described in [8] as the condition that "We must be entirely convinced that any proof of a theorem which the system certifies as correct should indeed be so." A straightforward way to formalize soundness would be to assume some sort of the semantics for $\mathcal{V}$, to take $\mathcal{M}$ powerful enough to express the notion of truth for the $\mathcal{V}$-formulas and to establish inside $\mathcal{M}$ a formal analogue of the statement

for every sentence φ if φ is provable then φ is true.

This approach would require a fairly strong $\mathcal{M}$. In particular, one needs to extend the language of $\mathcal{V}$ in order to write down formulas *"φ is true"*; by the well-known Tarski theorem there is no such formula in the language of $\mathcal{V}$ itself.

In fact, in the proof checking context, a verification system $\mathcal{V}$ deals with the true values of formal statements of an especially simple type, namely provable Δ_1 sentences $[\![t]\!]\varphi$. In this paper we assume that soundness of a verification system $\mathcal{V}$ means that all sentences $[\![t]\!]\varphi$ derivable in $\mathcal{V}$ are true.

Theorem 3. *1. $\mathcal{V}$ is consistent iff $\mathcal{V} \vdash [\![t]\!]\varphi$ implies $[\![t]\!]\varphi$,*
2. $\mathcal{V}$ suffices to establish 1.

Proof. If for all φ $\mathcal{V} \vdash [\![t]\!]\varphi$ implies $[\![t]\!]\varphi$, then no false sentences of the kind $[\![t]\!]\varphi$ is provable in $\mathcal{V}$, therefore $\mathcal{V}$ is consistent.

Suppose $\mathcal{V}$ is consistent and let $\mathcal{V} \vdash [\![t]\!]\varphi$. If $[\![t]\!]\varphi$ were false, then $\mathcal{V} \vdash \neg[\![t]\!]\varphi$, by Δ_1 completeness of $\mathcal{V}$. This leads to a contradiction in $\mathcal{V}$.

2. The straightforward formalization of the proof of 1 with the use of provable Δ_1 completeness of $\mathcal{V}$. ∎

Corollary 1. *Simple consistency of $\mathcal{V}$ is necessary and sufficient for soundness of a verification system $\mathcal{V}$.*

Now we will figure out what metatheory can establish stability.

Definition 10. *By $\mathcal{V}$* **is stable** *we understand the $\mathcal{V}$-formula which is the natural formalization of the stability property of $\mathcal{V}$. By* **implicit reflection rule is admissible in** $\mathcal{V}$ *we mean the natural formalization in the language of $\mathcal{V}$ of the property that $IRR(\mathcal{V})$ is admissible in $\mathcal{V}$; we will denote this formula*

$$\forall x(\Box\Box x \rightarrow \Box x).$$

Theorem 4.

$$\mathcal{V} \vdash \text{“}\mathcal{V} \textbf{ is stable} \leftrightarrow \textbf{implicit reflection rule is admissible in } \mathcal{V}\text{”}$$

Proof. The straightforward (though delicate) formalization of the proof of Theorem 1. ∎

Theorem 5. *Stability of an ω-consistent verification system is not provable in this system.*

Proof. By Theorem 4, stability is provable in $\mathcal{V}$ iff $\mathcal{V} \vdash \forall x(\Box\Box x \to \Box x)$. Let x is the code of $\bot$. Then $\mathcal{V} \vdash \Box\Box\bot \to \Box\bot$. By Löb's theorem, $\mathcal{V} \vdash \Box\bot$, which is impossible for an ω-consistent $\mathcal{V}$. ∎

It follows from the above that the minimal metatheory for soundness and implicit stability is

$$\mathcal{M} = \mathcal{V} + \mathit{Consis}(\mathcal{V}) + \forall x(\Box\Box x \to \Box x).$$

Theorem 6. *If $\mathcal{V}$ is correct with respect to the standard model of arithmetic then the metatheory for soundness and implicit stability strictly subsumes the first ω steps of the Turing progression.*

Proof. In order to establish $\mathcal{V}^c_\omega \subset \mathcal{M}$ consider the formulas $\Box^0\bot = \bot$, $\Box^{n+1}\bot = \Box(\Box^n\bot)$. First of all we note that under the assumptions made about $\mathcal{V}$ the formula $\mathit{Consis}(\mathcal{V}^c_n)$ is provably equivalent in $\mathcal{V}$ to $\neg\Box^{n+1}\bot$ (cf. [2]). Indeed, $\mathit{Consis}(\mathcal{V}^c_0)$ is $\mathit{Consis}(\mathcal{V})$, i.e. $\neg\Box\bot$. Then $\mathit{Consis}(\mathcal{V}^c_1)$ is a formula stating that $\mathcal{V} + \mathit{Consis}(\mathcal{V}) \nvdash \bot$, i.e. $\mathcal{V} + \neg\Box\bot \nvdash \bot$. This is equivalent to $\mathcal{V} \nvdash \neg\Box\bot \to \bot$ and $\mathcal{V} \nvdash \Box\bot$. Therefore, $\mathit{Consis}(\mathcal{V}^c_1)$ is equivalent to $\neg\Box\Box\bot$. Similar argument works for $n = 2, 3, 4, \ldots$.

Now we show how to derive all $\neg\Box^n\bot$, $n = 1, 2, 3, \ldots$ in $\mathcal{M}$. The case $n = 1$ is covered by the assumption that $\mathcal{M} \vdash \mathit{Consis}(\mathcal{V})$, which is equivalent to $\mathcal{M} \vdash \neg\Box\bot$, or $\mathcal{M} \vdash \Box\bot \to \bot$ For $n = 2$ put $x = \bot$ in $\forall x(\Box\Box x \to \Box x)$. Then $\mathcal{M} \vdash \Box\Box\bot \to \Box\bot$. Since we have already had $\mathcal{M} \vdash \Box\bot \to \bot$, we conclude that $\mathcal{M} \vdash \Box\Box\bot \to \bot$, i.e. $\mathcal{M} \vdash \neg\Box\Box\bot$. A similar argument works for $n = 3, 4, 5, \ldots$. Thus

$$\mathcal{V}^c_\omega \subset \mathcal{M}.$$

Now we will check that $\mathcal{V}^c_\omega \neq \mathcal{M}$. Suppose

$$\mathcal{V}^c_\omega \vdash \mathit{Consis}(\mathcal{V}) \wedge \forall x(\Box\Box x \to \Box x).$$

By the compactness argument, there is a natural number n such that

$$\mathcal{V}^c_n \vdash \mathit{Consis}(\mathcal{V}) \wedge \forall x(\Box\Box x \to \Box x).$$

Since $\mathcal{V}^c_\omega \subset \mathcal{M}$, $\mathcal{M}$ proves the consistency of $\mathcal{V}^c_n$. Therefore

$$\mathcal{V}^c_n \vdash \mathit{Consis}(\mathcal{V}^c_n),$$

which is impossible by the second Gödel incompleteness theorem for $\mathcal{V}^c_n$. ∎

6 Metamathematical Cost of Implicit Reflection

In an ω-consistent verification system $\mathcal{V}$ the rule of implicit reflection $IRR(\mathcal{V})$ is admissible, i.e. $\mathcal{V} \vdash \Box\varphi$ yields $\mathcal{V} \vdash \varphi$ for any formula φ. The most simple formalization of the admissibility property is the scheme $\Box\Box\varphi \rightarrow \Box\varphi$, where $\Box\psi$ stands for the formula of provability of ψ in $\mathcal{V}$. A general procedure of incorporating implicit reflection rule into a verification system $\mathcal{V}$ may be presented by the following *reflection tower* of extensions of $\mathcal{V}$ (cf. [12], [1], [11]):

$$\mathcal{V}^r_0 = \mathcal{V}, \quad \mathcal{V}^r_{\alpha+1} = \mathcal{V}^r_\alpha + IRR(\mathcal{V}^r_\alpha), \quad \mathcal{V}^r_\gamma = \bigcup_{\beta \prec \gamma} \mathcal{V}^r_\beta \textit{ for a limit ordinal } \gamma.$$

For the sake of simplicity we assume in this section that $\mathcal{V}$ is sound with respect to the standard model of arithmetic.

In this section we will try to figure out what natural metatheory is able to establish the admissibility of all the reflection rules from the reflection tower.

Definition 11. *Implicit reflection principle $IRP(\mathcal{V})$ for a given system $\mathcal{V}$ is the scheme of formulas*

$$\{\Box\varphi \rightarrow \varphi \mid \varphi \textit{ is a sentence of } \mathcal{V}\}.$$

Let us consider *Feferman's progression* of extensions of $\mathcal{V}$ by the implicit reflection principles ([10]):

$$\mathcal{V}^p_0 = \mathcal{V}, \quad \mathcal{V}^p_{\alpha+1} = \mathcal{V}^p_\alpha + IRP(\mathcal{V}^p_\alpha), \quad \mathcal{V}^p_\gamma = \bigcup_{\beta \prec \gamma} \mathcal{V}^p_\beta \textit{ for a limit ordinal } \gamma.$$

The system $\mathcal{V}^p_1$ proves admissibility of implicit reflection in $\mathcal{V}^r_0$, i.e. the scheme of formulas $\Box\Box\varphi \rightarrow \Box\varphi$. In addition $\mathcal{V}^p_1 \subset \mathcal{V}^r_1$, since every instance of the rule $\Box\varphi/\varphi$ in a proof in $\mathcal{V}^r_1$ can be emulated by the axiom $\Box\varphi \rightarrow \varphi$. Moreover, the inclusion $\mathcal{V}^p_1 \subset \mathcal{V}^r_1$ can be established in $\mathcal{V}$. Iterating this argument one can show that $\mathcal{V}^p_{\alpha+1}$ is the theory capable of establishing admissibility of the implicit reflection rule for $\mathcal{V}^r_\alpha$.

How bad really is the reflection tower for $\mathcal{V}$? The natural metatheory capable of verifying the whole reflection tower is the limit of Feferman's progression $\mathcal{V}^p_\alpha$ for all constructive ordinals α.

Proposition 1. *([10]) The limit of $\mathcal{V}^p_\alpha$ for all constructive ordinals α equals*

$$\mathcal{V} \ + \ \textit{all true } \Pi_1\textit{-sentences.}$$

It follows from the above that the natural metatheory for the reflection tower is not computably enumerable, and could not possibly be verified by any sound mathematical means. It contains, for example, the consistency statements for all consistent axiomatic theories, among them $Consis(\mathcal{ZF})$ (provided $\mathcal{ZF}$ is consistent).

In the next section we describe the explicit reflection, which is internally verifiable and thus circumvents the reflection tower.

7 Explicit Reflection for Verification Systems

An alternative way to represent provability in a logical setting has been suggested in [3] – [6], where a basic theory of *explicit provability* was developed. The key idea of this approach is to switch from the uniform but implicit presentation of provability of φ as $\Box\varphi$ to a presentation of provability of φ by a certain family of explicit *proof terms* $[\![t]\!]\varphi$ (i.e. $Proof(t, \ulcorner\varphi\urcorner)$) depending on the context. As it was shown in [5] and [6], every propositional property of the provability operator (to the extend of the modal logic **S4**) can be represented by the family of finitely generated proof terms. Within this explicit provability approach some old problems in theoretical logic were solved. In particular, explicit provability provided the intended provability semantics for intuitionistic logic by formalizing Brouwer-Heyting-Kolmogorov semantics (the problem was open since 1930) and for the modal logic **S4** (was open since 1933).

In this paper we introduce a reflection mechanism based on explicit provability. This mechanism could help to avoid metamathematical costs of using the implicit reflection without restricting real verification capacities of a system.

Theorem 7. *For every sentence φ such that $\mathcal{V} \vdash \varphi$ there is a ground term t of $\mathcal{V}$ such that $\mathcal{V} \vdash [\![t]\!]\varphi$.*

Proof. Given $\mathcal{V} \vdash \varphi$ let D be a derivation of φ in $\mathcal{V}$. Let $t{=}\ulcorner D\urcorner{=}rep(D)$. By the assumptions on the function *rep*, $[\![t]\!]\varphi$ holds. Since $\mathcal{V}$ is able to represent all true Δ_1 facts, $\mathcal{V} \vdash [\![t]\!]\varphi$.

Definition 12. *The* explicit reflection principle *$ERP(\mathcal{V})$ is the scheme of formulas $[\![t]\!]\varphi \to \varphi$ for all sentences φ and all ground terms t.*

Theorem 8. (Provability of explicit reflection [3]). *For any ground term t and formula φ*

$$\mathcal{V} \vdash [\![t]\!]\varphi \to \varphi.$$

Proof. We give a constructive proof of this lemma which delivers an algorithm for constructing a derivation of $[\![t]\!]\varphi \to \varphi$ in $\mathcal{V}$ given φ and t. First of all, by the proof checking procedure we calculate the truth value of $[\![t]\!]\varphi$. If this value is *TRUE*, then the ground term t represents a derivation of φ, from which by a straightforward reconstruction, we obtain the proof of $[\![t]\!]\varphi \to \varphi$. If the proof checker on $[\![t]\!]\varphi$ returns *FALSE*, then by the corresponding procedure we get the proof of $\neg[\![t]\!]\varphi$ in $\mathcal{V}$. From that by the straightforward transformation, we get the proof of $[\![t]\!]\varphi \to \varphi$. ∎

Corollary 2. *There is an algorithm which given a formula φ and a ground term t returns the ground term p such that*

$$\mathcal{V} \vdash [\![p]\!]([\![t]\!]\varphi \to \varphi).$$

Definition 13. *A rule Γ/φ is explicitly verifiable in $\mathcal{V}$ if there is a total computable function f such that $\mathcal{V} \vdash [\![y]\!]\Gamma \rightarrow [\![f(y)]\!]\varphi$.*

Theorem 9.

1. *Every derivable and every provable rule is explicitly verifiable.*
2. *Every explicitly verifiable rule is verifiable.*
3. *Every explicitly verifiable rule is admissible.*

Proof. 1. There is a straightforward function behind every internal rule Δ/ψ which calculates the code of a proof of ψ given the codes of proofs of Δ. A natural formalization of this function in $\mathcal{V}$ gives a term f such that $\mathcal{V} \vdash [\![y]\!]\Delta \rightarrow [\![f(y)]\!]\psi$. The same holds for all derivable rules. If a rule Γ/φ is provable then $\mathcal{V} \vdash \Gamma \rightarrow \varphi$. By explicit formalization (Theorem 7), $\mathcal{V} \vdash [\![t]\!](\Gamma \rightarrow \varphi)$ for some ground term t. Let "$\cdot$" be a total and computable "application" function on proof codes, specified by the condition

$$\mathcal{V} \vdash [\![x]\!](\varphi \rightarrow \psi) \rightarrow ([\![y]\!]\varphi \rightarrow [\![x \cdot y]\!]\psi)$$

(cf. [5], [6]). In particular,

$$\mathcal{V} \vdash [\![t]\!](\Gamma \rightarrow \varphi) \rightarrow ([\![x]\!]\Gamma \rightarrow [\![t \cdot x]\!]\varphi),$$

which yields $\mathcal{V} \vdash [\![x]\!]\Gamma \rightarrow [\![t \cdot x]\!]\varphi$.

2. From $\mathcal{V} \vdash [\![y]\!]\Gamma \rightarrow [\![f(y)]\!]\varphi$ it easily follows that $\mathcal{V} \vdash \Box\Gamma \rightarrow \Box\varphi$.

3. Let $\mathcal{V} \vdash [\![y]\!]\Gamma \rightarrow [\![f(y)]\!]\varphi$ and suppose that $\mathcal{V} \vdash \Gamma$. By Theorem 7, $\mathcal{V} \vdash [\![t]\!]\Gamma$ for some ground term t. Therefore $\mathcal{V} \vdash [\![t]\!]\Gamma \rightarrow [\![f(t)]\!]\varphi$ and $\mathcal{V} \vdash [\![f(t)]\!]\varphi$. By Theorem 8, $\mathcal{V} \vdash [\![f(t)]\!]\varphi \rightarrow \varphi$. Thus $\mathcal{V} \vdash \varphi$ ∎

Definition 14. *The explicit reflection rule ERR($\mathcal{V}$) is the rule $[\![t]\!]\varphi/\varphi$ for all ground terms t and all sentences φ.*

Theorem 10. *The explicit reflection rule ERR($\mathcal{V}$) is explicitly verifiable in $\mathcal{V}$.*

Proof. By Theorem 8, $\mathcal{V} \vdash [\![p]\!]([\![t]\!]\varphi \rightarrow \varphi)$ for some ground term p. By the same argument about computable "application" as in the proof of the previous theorem,

$$\mathcal{V} \vdash ([\![y]\!]\,[\![t]\!]\varphi \rightarrow [\![p \cdot y]\!]\varphi).$$

∎

Corollary 3. *The explicit reflection rule ERR($\mathcal{V}$) is admissible for every verification system $\mathcal{V}$.*

Definition 15. *An extension $\mathcal{V}'$ of $\mathcal{V}$ is verifiably equivalent to $\mathcal{V}$ if there is a computable function g of $\mathcal{V}$ such that $\mathcal{V} \vdash [\![x]\!]'\psi \rightarrow [\![g(x)]\!]\psi$, where $[\![x]\!]'\psi$ stands for the formula "x is a proof of ψ in $\mathcal{V}'$. In other words, for a verifiably equivalent extension $\mathcal{V}'$ there is an algorithm that transforms proofs in $\mathcal{V}'$ into proofs of the same facts in $\mathcal{V}$.*

Theorem 11. *An extension of a verification system by an explicitly verified rule is verifiably equivalent to the original system.*

Proof. Let a rule Γ/φ be explicitly verifiable in a verification system $\mathcal{V}$, i.e. there is a computable function f such that $\mathcal{V} \vdash [\![y]\!]\Gamma \rightarrow [\![f(y)]\!]\varphi$. Let $\mathcal{V}'$ be $\mathcal{V} + \Gamma/\varphi$. The function $g(x)$ works as follows. It travels along the proof tree in $\mathcal{V}'$ coded by x and calculates the code of a proof tree in $\mathcal{V}$ of the same sentence (sequent). If the observed node is a leaf node, then it corresponds to an axiom of $\mathcal{V}'$, which is an axiom of $\mathcal{V}$ as well. In this situation g does not change the the proof at all.

Let the observed node correspond to an application of an internal rule Δ/θ, and let $\boldsymbol{u}$ be the values of g on the predecessors of the current node, i.e. $\mathcal{V} \vdash [\![\boldsymbol{u}]\!]\Delta$. By Theorem 9, there is a computable function h such that $\mathcal{V} \vdash [\![\boldsymbol{y}]\!]\Delta \rightarrow [\![h(\boldsymbol{y})]\!]\theta$. Substituting u for y we derive $[\![h(\boldsymbol{u})]\!]\theta$ in $\mathcal{V}$. Let g map the observed node to $h(\boldsymbol{u})$.

Let the observed node correspond to an application of the new rule Γ/φ, and let $\boldsymbol{v}$ be the values of g on the predecessors of this node, i.e. $\mathcal{V} \vdash [\![\boldsymbol{v}]\!]\Gamma$. By the conditions of the theorem $\mathcal{V} \vdash [\![\boldsymbol{y}]\!]\Gamma \rightarrow [\![f(\boldsymbol{y})]\!]\varphi$. Substitute v's for y's, conclude that $\mathcal{V} \vdash [\![f(\boldsymbol{v})]\!]\varphi$ and let g map the observed node to $f(\boldsymbol{v})$.

Eventually, at the root node of the $\mathcal{V}'$-proof (coded by) x the function g returns the code of a $\mathcal{V}$-proof of the formula (sequent) previously proven by x. ∎

8 Practical Suggestions

As one can see, explicit reflection avoids some of the troubles inherent in implicit reflection. Here is the list of practical suggestions for the designers of verification systems.

1. **Proof Checking.** Explicit reflection is used by default in proof checking when one concludes that $\mathcal{V}$ has verified a fact φ given that $\mathcal{V} \vdash [\![t]\!]\varphi$ for some proof code t. This scheme is theoretically correct and does not contain any extra hidden metamathematical costs. The use of explicit reflection here should be acknowledged.

2. **Extendable Verification Systems.** Here the use of explicit reflection may be twofold. Firstly, it appears in the *assertion insertion mode* (cf. [8]), when it is established that $\mathcal{V} \vdash [\![t]\!]\varphi$ and then φ is stored as a verified fact (i.e. a new axiom) of $\mathcal{V}$. We have nothing specific to add here, since this mode as presented above (and in [8]) already agrees with the explicit reflection recommendations. Secondly, the explicit reflection appears in the *rule insertion mode*, when Γ/φ is verified in $\mathcal{V}$ and then added to $\mathcal{V}$ as a new inference rule. The explicit reflection suggests verifying the rule Γ/φ in $\mathcal{V}$ explicitly, i.e. by constructing a computable function f such that $\mathcal{V} \vdash [\![y]\!]\Gamma \rightarrow [\![f(y)]\!]\varphi$. By doing this we guarantee that the resulting extension is verified in the old system without any hidden meta assumptions.

> If the rule insertion mode uses explicit verification only, then there is no need to have a special built-in reflection mechanism: provable stability of the system is preserved by explicit verification (Theorem 11).

Interestingly enough, there are substantial classes of verification systems where the implicit verification in a certain sense yields the explicit one. For example, in many intuitionistic systems $\mathcal{V} \vdash \Box\Gamma \rightarrow \Box\varphi$ implies $\mathcal{V} \vdash [\![y]\!]\Gamma \rightarrow [\![f(y)]\!]\varphi$ for some computable function f (cf. [17]). However, the proof of this fact itself cannot be formalized in $\mathcal{V}$ and its use in the rule insertion mode leads to some sort of a reflection tower. Therefore, ever for the constructive systems the practical suggestion would be to use the explicit verification, i.e. to establish $\mathcal{V} \vdash [\![y]\!]\Gamma \rightarrow [\![f(y)]\!]\varphi$ directly rather than to prove $\mathcal{V} \vdash \Box\Gamma \rightarrow \Box\varphi$ and then to apply a general theorem of obtaining the explicit verification from the implicit one; this involves some hidden and potentially high metamathematical costs.

3. **Advanced systems with built-in reflection mechanisms.** There is a number of systems which have or intend to have such mechanisms. The paper [11] mentions several of them: **FOL**, **NQTHM**, **HOL** and **Nuprl**. At least one more is coming: **MetaPrl** at Cornell University. Probably more systems will join this set since reflection arguments are often used in mathematical and common reasoning (cf. Section 4). The existing implicit reflection mechanisms in these systems lead to unnecessary metamathematical costs (cf. Section 6). For such systems the idea of having explicit reflection (perhaps, along with the implicit one) might be seriously considered, because the explicit reflection can be added to a system without any extra metamathematical assumptions at all (Theorem 10).

Acknowledgements

I am indebted to Robert Constable for attracting my attention to the problem of reflection in verification systems and for support during my work on this paper. I am also grateful to Stuart Allen, Anil Nerode, Elena Nogina and Vaughan Pratt for valuable suggestions and criticism. Many thanks to Martin Davis for fruitful discussion and for sending me a copy of his paper.

References

1. Allen, S., Constable R., Howe D., Aitken W.: The Semantics of Reflected Proofs. In: Proceedings of the Fifth Annual Symposium on Logic in Computer Science, Los Alamitos, CA, USA, IEEE Computer Society Press (1990) 95-107.
2. Artemov, S.: Extensions of Theories by the Reflection Principles and the Corresponding Modal Logics. Ph.D. Thesis (in Russian). Moscow (1979)
3. Artemov, S., Strassen, T.: The Basic Logic of Proofs. In.: Lecture Notes in Computer Science. Vol. 702. Springer-Verlag (1993) 14-28.
4. Artemov, S.: Logic of Proofs. Annals of Pure and Applied Logic **67** (1994) 29-59
5. Artemov, S.: Operational Modal Logic. Technical Report MSI 95-29, Cornell University (1995)
6. Artemov, S.: Explicit Provability: the Intended Semantics for Intuitionistic and Modal Logic. Technical Report CFIS 98-10, Cornell University (1998)
7. Boolos, G.: The Unprovability of Consistency: An Essay in Modal Logic. Cambridge University Press (1979)

8. Davis, M., Schwartz, J.: Metamathematical Extensibility for Theorem Verifiers and Proof Checkers. Computers and Mathematics with Applications **5** (1979) 217-230
9. Feferman, S.: Arithmetization of Metamathematics in a General Setting. Fundamenta Mathematicae **49** (1960) 35-92
10. Feferman, S.: Transfinite Recursive Progressions of Axiomatic Theories. Journal of Symbolic Logic **27** (1962) 259-316
11. Harrison, J.: Metatheory and Reflection in Theorem Proving: A Survey and Critique. University of Cambridge (1995) `http://www.dcs.glasgow.ac.uk/~tfm/hol-bib.html\#H`
12. Knoblock, T., Constable, R.: Formalized Metareasoning in Type Theory. In: Proceedings of the First Annual Symposium on Logic in Computer Science. Cambridge, MA, USA, IEEE Computer Society Press (1986) 237-248
13. Kreisel, G., Levy, A.: Reflection Principles and Their Use for Establishing the Complexity of Axiomatic Systems. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik **14** (1968) 97-142
14. Pollack, R.: On Extensibility of Proof Checkers. Lecture Notes in Computer Science, Vol. 996. Springer-Verlag, Berlin Heidelberg New York (1995) 140-161
15. Smorynski, C.: The Incompleteness Theorems. In: Barwise, J (ed.): Handbook of Mathematical Logic, Vol. 4. North-Holland, Amsterdam (1977) 821-865
16. C. Smorynski, C.: Self-Reference and Modal Logic. Springer-Verlag, Berlin (1985)
17. A.S. Troelstra, A.S., van Dalen, D.: Constructivism in Mathematics. An Introduction, Vol. 1, Amsterdam; North Holland (1988)
18. Turing, A.: Systems of Logics Based on Ordinals. Proceedings of the London Mathematical Society, Ser. 2, Vol. 45 (1939) 161-228

System Description: KIMBA, A Model Generator for Many-Valued First-Order Logics

Karsten Konrad[1] and D. A. Wolfram[2]

[1] Fachbereich Informatik, Universität des Saarlandes
[2] Department of Computer Science, The Australian National University

1 Overview

KIMBA is the first model generation program which implements a semi-decision procedure for finite satisfiability of first-order logics with finitely many truth values. The procedure enumerates the finite models of its input and can be used to compute efficiently domain minimal models whose positive part is minimal in size. KIMBA has been implemented in the constraint logic programming language Oz [6] and is based on a tableaux calculus that translates deduction problems into Constraint Satisfaction Problems (CSPs). The constraint propagators needed to solve these problems are realized as concurrent procedures that can make use of Oz's built-in capabilities for solving CSPs.

We describe the current prototype focusing on the method for generating and solving CSPs.

2 Theoretical Background

Model generators are automated deduction systems that prove the satisfiability of logical theories by generating models [3]. KIMBA is a model generator for many-valued logics which semi-decides the finite satisfiability of first-order theories and decides the satisfiability of propositional ones. Its proof procedure is a generalisation of Hähnle's translation of deduction problems into mixed-integer programming [2]. Hähnle's method constructs *constraint tableaux* for arbitrary propositional theories and produces sets of mixed-integer inequations which can be solved by an external constraint solver. The set of inequations generated is linear in size to that of the input theory and the solutions produced by the constraint solver are models of this input. The method does not require clause normalisation, an advantage in practice where suitable normalisation routines may not be available for the given logic. It also avoids the combinatorial explosion of tableau branches in certain many-valued logics where the branching factor of propositional rules can equal the number of truth values. However, Hähnle's method cannot be used for first-order model generation, and the mixed-integer inequations generated quickly lead to intractable CSPs.

The model generator KIMBA generalises Hähnle's idea to a constraint tableau system whose propagation mechanism can be tailored to the underlying logic.

KIMBA accepts higher-order specifications and makes use of Oz's built-in propagators and constraint solving system for integer variables. It translates quantified formulas into constraints over finite nonempty domains of constants. In the case of first-order quantification, the domain used is a finite set of individual constants including those that occur in the problem specification. Using iterative deepening over the size of this domain, the constraint solving process in Oz enumerates all finite first-order Herbrand models of the input up to a renaming of individual constants [5]. Hence, KIMBA provides a semi-decision procedure for finite satisfiability for conventional multi-valued theorem proving systems such as ${}_3T^AP$ [1] which do not halt in general on a satisfiable input theory, even when the input has a finite model.

Quantifications of the form $\forall X_\alpha\ P(X)$ or $\exists X_\alpha\ P(X)$, with X_α being a higher-order variable of type α, are translated to constraints over a given signature Σ_α of higher-order constants of type α. This actually implements a restricted form of higher-order model generation [5] and is a compact way to formalise properties of finite sets of predicates and functions.

3 Building Constraint Tableaux

Tableaux expansion in KIMBA starts with a set of pairs of the form V_{F}: F where F is an initial closed formula and V_{F} is a finite domain integer variable whose value is associated with the truth value of F. Functions are translated into relations in a preprocessing step. A formula F can be signed by the user with a truth value v indicating that F must be interpreted as v in all models. The value v is an integer that can range from 0 to n where n is the highest possible truth value in the many-valued logic considered. Initial formulas F are signed by default with n, and their sign immediately determines their truth variables V_{F}.

$$\frac{V_{\mathsf{F}}\colon \mathsf{F}_1 \vee \mathsf{F}_2}{\begin{array}{c} V_{\mathsf{F}_1}\colon \mathsf{F}_1 \\ V_{\mathsf{F}_2}\colon \mathsf{F}_2 \\ \mathbf{D}(V_{\mathsf{F}_1}, V_{\mathsf{F}_2}, V_{\mathsf{F}}) \end{array}}\ \vee \qquad \frac{V_{\neg\mathsf{F}}\colon \neg\mathsf{F}}{\begin{array}{c} V_{\mathsf{F}}\colon \mathsf{F} \\ \mathbf{N}(V_{\mathsf{F}}, V_{\neg\mathsf{F}}) \end{array}}\ \neg \qquad \frac{V\colon \forall x_\alpha\ \mathsf{F}}{\begin{array}{c} V_{\mathsf{F},c_1}\colon [c_1/x]\mathsf{F} \\ \vdots \\ V_{\mathsf{F},c_k}\colon [c_k/x]\mathsf{F} \\ \mathbf{A}(\{V_{\mathsf{F},c_1}, \ldots, V_{\mathsf{F},c_k}\}, V) \end{array}}\ \forall$$

Fig. 1. Three of KIMBA's tableaux rules for generating constraints

Figure 1 shows three of the rules that decompose the logical structure of the input. The rules for $\wedge, \Rightarrow, \Leftrightarrow$ and $\exists$ are analogous. The $\vee$-rule introduces two new pairs V_{F_1}: F_1 and V_{F_2}: F_2 for the components of a disjunction and adds a constraint $\mathbf{D}(V_{\mathsf{F}_1}, V_{\mathsf{F}_2}, V_{\mathsf{F}})$ which constrains the values of the integer variables of the disjunctive formula and its components. The $\neg$-rule is very similar to $\vee$, but introduces only one new pair for the formula in the scope of the negation, and a constraint $\mathbf{N}(V_{\mathsf{F}}, V_{\neg\mathsf{F}})$. The constraint predicates **D** and **N** are implemented as concurrent procedures in Oz. They must be specified separately for each logic as we shall below for a 3-valued version of **N**.

The $\forall$-rule adds pairs $\{V_{\mathsf{F},c_1}\colon [c_1/x]\mathsf{F}, \ldots, V_{\mathsf{F},c_k}\colon [c_k/x]\mathsf{F}\}$ for all instantiations $[c_i/x_\alpha]\mathsf{F}$ of $\forall x_\alpha\ \mathsf{F}$ that can be made with the current domain $\Sigma_\alpha = \{c_1, \ldots, c_k\}$ of constants of type α. The truth variable V associated with $\forall x_\alpha\ \mathsf{F}$ is then constrained by $\mathbf{A}(\{V_{\mathsf{F},c_1}, \ldots, V_{\mathsf{F},c_k}\}, V)$. Again, the operational semantics of **A** depends on the logic considered. For instance, in classical logic where 1 denotes truth and 0 denotes falsity, the constraint **A** can be defined as follows:

$$\mathbf{A}(\{V_{\mathsf{F},c_1}, \ldots, V_{\mathsf{F},c_k}\}, V) \ ::= \ (V_{\mathsf{F},c_1} + \ldots + V_{\mathsf{F},c_k} = k) \Leftrightarrow V = 1$$

This means that the truth value V of a universally quantified formula is 1 iff the sum $V_{\mathsf{F},c_1} + \ldots + V_{\mathsf{F},c_k}$ of the truth values of the formula's instantiations is equal to k. This projection of the validity of arithmetic constraints into integer variables that denote truth values is called *constraint reification.* Oz provides a constraint language for finite-domain integer variables that allows us to translate quantified formulas directly into reified constraints like the one above.

Unlike conventional tableaux systems, Kimba's calculus does not split the current tableau branch during tableau construction. The multiple decomposition of identical formulas in different branches, a well-known weak point in many tableaux systems, is avoided.

4 Solving Constraints in Oz

Defining a logic in Kimba means implementing propagator procedures for the logical connectives and the quantifiers. Each propagator defines the semantics of its associated connective or quantifier. Kimba's generic translation of deduction into Oz constraints allows us to design dedicated, optimised propagators for each logic. Oz itself already provides an efficient set of propagators for the standard connectives in classical logic. Additionally, the prototype implements a 3-valued logic for partiality [4]. Below are two operationally equivalent propagator procedures for negation (**N**) in our 3-valued logic. While the second procedure appears to be more elegant, it introduces an arithmetic equation, as in Hähnle's approach, which can be harder to process than the simple symbolic propagation in the first procedure.

```
proc {N1 V1 V2}
   [V1 V2] ::: 0#2
   thread
      if V1 = 0 then V2 = 2
      [] V1 = 1 then V2 = V1
      [] V1 = 2 then V2 = 0
      [] V2 = 0 then V1 = 2
      [] V2 = 1 then V1 = V2
      [] V2 = 2 then V1 = 0
      end
   end
end
```

```
proc {N2 V1 V2}
   [V1 V2] ::: 0#2
   {FD.minus 2 V1 V2}
end
```

Procedure N1 first restricts its two parameters to values between 0 ("false") and 2 ("true"), where 1 is used for the truth value "undefined". Then it starts a concurrent process which waits for one of these parameters to be determined and determines the other one accordingly. Procedure N2 does the same, simply by constraining the values of V_1 and V_2 by the equation $V_2 = 2 - V_1$.

Propagation is concurrent, so the process of determining integer variables V_{F} can partly take place during tableau construction. An unsatisfiable tableau branch can often be closed immediately when a variable is constrained inconsistently. After a branch is saturated and the initial propagation has been completed, undetermined variables are provisionally restricted further, and the next iteration of propagation begins. As an heuristic, KIMBA restricts first those variables whose value affects as many other variables and constraints as possible. It also minimises the number of positive literals that are validated in the model at the same time. The process of variable distribution and propagation is repeated until all integer variables have a unique value. If it is not possible to assign each formula a unique integer, then the tableau is unsatisfiable and KIMBA restarts model search by extending the current domain of individuals with a newly generated constant and by building up a new constraint tableau.

The iterative deepening search for models will always find those models first whose domain of individuals is minimal. Additionally, search can be bounded by the number of literals that are validated. This implements another weak form of minimal model reasoning. By combining this restriction with domain minimality, KIMBA efficiently computes those domain minimal models of the input whose presentation as a finite set of positive literals is as short as possible.

5 System Performance

The table below shows KIMBA's performance on some selected problems from the the TPTP library [8]. Times were taken on a Sparc Ultra 1.

Problem	PUZ001-1	PUZ005-1	PUZ017-1	PUZ031-1	MSC007-2 (5)	MSC007-2 (8)
Time	0.3s	4.6s	0.7s	1.6s	0.3s	107.3s

The original TPTP formalisations are clausal theories that have been re-formalised for KIMBA using first- and higher-order specifications in classical 2-valued logic. The TPTP formalisations of the logical puzzles PUZ001-1 to PUZ031-1 are unsatisfiable. In contrast to this, the KIMBA formalisation for each puzzle is satisfiable, and each model produced corresponds to a solution. Cook's pigeon-hole problem, MSC007-2, is a classical benchmark proof problem for propositional theorem provers. The table shows the performance of Kimba on problem instantiations with five and eight holes.

PUZ017-1 is very hard for most theorem provers because its first-order clause representations leads to a large search space. Kimba's exceptionally good performance can be explained by an elegant higher-order formalisation. For instance, the specification *every job is held by at least one person* in PUZ017-1 has a natural formalisation in Kimba as follows:

$$\forall J_{\iota\mapsto o}\ Job_{(\iota\mapsto o)\mapsto o}(K) \Rightarrow \exists x_{\iota} Person_{\iota\mapsto o}(x) \wedge J(x)$$

With this form of quantification, Kimba keeps entities of different type separately. In a first-order logic, all quantified entities must be individuals. This usually leads to a larger than necessary domain for which model generation and theorem proving becomes exponentially more complex.

The Kimba prototype does not employ any of the sophisticated data structures and low-level optimisations that can be found in other comparable model generation programs such as FINDER [7]. Hence, Kimba is not yet efficient enough for instance for quasi-group existence problems where our translation of deduction into constraint solving does not effectively restrict large search spaces.

References

1. B. Beckert, R. Hähnle, P. Oel, and M. Sulzmann. The tableau-based theorem prover $_3T^AP$, version 4.0. In M. McRobbie and J. Slaney, editors, *Proc., 13th International Conference on Automated Deduction (CADE), New Brunswick, NJ, USA*, LNCS 1104. Springer, 1996.
2. R. Hähnle. Many-valued logic and mixed integer programming. *Annals of Mathematics and Artificial Intelligence*, 12(3,4):231–264, Dec. 1994.
3. R. Hasegawa. Model generation theorem provers and their applications. In L. Sterling, editor, *Proceedings of the 12th International Conference on Logic Programming*, pages 7–8, Cambridge, MA, USA, June13–18 1995. MIT Press.
4. M. Kerber and M. Kohlhase. A mechanization of strong Kleene logic for partial functions. SEKI-Report SR-93-20 (SFB), Universität des Saarlandes, Saarbrücken, 1993.
5. K. Konrad and D. A. Wolfram. Finite model generation for many-valued higher-order logics. Forthcoming, 1999.
6. Programming Systems Lab Saarbrücken, 1998. Oz Webpage: http://www.ps.uni-sb.de/oz/.
7. J. Slaney. FINDER (Finite Domain Enumerator): Notes and guide. Technical Report TR-ARP-1/92, Australian National University Automated Reasoning Project, Canberra, 1992.
8. C. Suttner and G. Sutcliffe. The TPTP problem library (TPTP v2.2.0). Technical Report 97-04, Department of Computer Science, James Cook University, Townsville, Australia, 1998.

System Description: Teyjus—A Compiler and Abstract Machine Based Implementation of λProlog

Gopalan Nadathur and Dustin J. Mitchell

Department of Computer Science
University of Chicago
Ryerson Hall, 1100 E 58th Street, Chicago, IL 60637
`gopalan@cs.uchicago.edu, djmitche@cs.uchicago.edu`

Abstract. The logic programming language λProlog is based on the intuitionistic theory of *higher-order hereditary Harrop formulas*, a logic that significantly extends the theory of Horn clauses. A systematic exploitation of features in the richer logic endows λProlog with capabilities at the programming level that are not present in traditional logic programming languages. Several studies have established the value of λProlog as a language for implementing systems that manipulate formal objects such as formulas, programs, proofs and types. Towards harnessing these benefits, methods have been developed for realizing this language efficiently. This work has culminated in the description of an abstract machine and compiler based implementation scheme. An actual implementation of λProlog based on these ideas has recently been completed. The planned presentation will exhibit this system—called *Teyjus*—and will also illuminate the metalanguage capabilities of λProlog.

1. Introduction. In work going back over a decade, Miller, Nadathur and colleagues have studied the proof-theoretic foundations of logic programming. A result of these investigations is the establishment of the intuitionistic theory of *higher-order hereditary Harrop formulas* as a suitable basis for this paradigm of programming [8]. This class of formulas enriches that of Horn clauses—the traditional basis for logic programming—with the possibilities of quantifying over function and (certain occurrences of) predicate variables, of explicitly representing binding in terms and of using a fuller complement of connectives and quantifiers. The λProlog language [13] is based on the logic of higher-order hereditary Harrop formulas. By systematically exploiting the new features in the underlying logic, λProlog provides support at the programming level for capabilities such as higher-order programming, polymorphic typing, scoping over names and procedures, modular programming, abstract data types and the use of lambda terms as data structures. Much research has been conducted in recent years towards fully understanding the usefulness of these additions to logic programming. Two aspects that have received special attention are the availability of lambda terms for representing objects and of a suitable set of primitives for manipulating such representations. These features enable λProlog to support the notion of *higher-order abstract syntax* [18] that is a profitable way to view the syntax of objects whose structure involves binding. Several detailed studies (e.g. [3], [5], [7]) have indicated the utility of λProlog in building systems that manipulate formal objects such as formulas, programs, proofs and types, making it comparable to other recently proposed metalanguages and logical frameworks such as Coq [1], Elf [17] and Isabelle [16]. Fuelled by these applications, Nadathur and colleagues have investigated methods for providing an efficient and robust realization of the language. This work has culminated recently in an actual implementation called *Teyjus*. We provide some insight here into the overall implementation scheme and its realization.

2. The λProlog Abstract Machine. An integral part of the implementation scheme is an abstract machine that is capable of realizing the operations that arise in typical λProlog programs efficiently. As with other logic programming languages, unification and backtracking are intrinsic to λProlog and the Warren Abstract Machine [19] provides a basic structure for treating these aspects well. However, an extensive embellishment of this framework is needed for realizing the following additional features satisfactorily:

– The language contains primitives that can alter the name space and the definitions of procedures in the course of execution. This means, in particular, that unification has to pay attention to changing signatures and that the solution to each (sub)goal has to be relativized to a specific program context.
– Lambda terms are used in λProlog as *data structures*. A representation must therefore be provided for these terms that permits their structures to be examined and compared in addition to supporting reduction operations efficiently.
– Higher-order unification is used in an intrinsic way in the language. This operation has a branching character that must be supported. Further, it is sometimes preferable to delay the solution to unification problems and so a good method is needed for transmitting these across computation steps.
– In addition to having a role in determining program correctness, types are relevant to the dynamic behavior of programs. A sensible scheme must therefore be included for carrying these along at run-time.
– Programming in the language is done relative to modules. In realizing this feature, it is necessary to support certain operations for composing modules. Moreover, a mechanism must be provided for periodically adding and removing code depending on which modules are in use.

The abstract machine that has been developed includes devices for treating all these aspects well. The solution to the problem of changing signatures is based on an elegant scheme for tagging constants and variables and using these tags in unification [9]. To realize changing program contexts, a fast method has been designed for adding and removing code that is capable also of dealing with backtracking [11]. The code that needs to be added may sometimes contain global variables and this possibility has been dealt with by an adaptation to logic programming of the idea of a closure. To facilitate a sensible representation of lambda terms, a new notation has been designed for these terms that utilizes the scheme of de Bruijn for eliminating names and that additionally supports an incremental calculation of reduction substitutions [15]. This notation has then been deployed systematically in the low-level steps contained in the abstract machine [10]. A method has been devised for the purpose of representing suspended unification problems [12] that has several interesting features. For example, based on the observation that new such problems arise out of incremental changes to old ones, the scheme is designed to support sharing while still making it possible to rapidly reinstate a previous unification problem upon backtracking. The treatment of higher-order unification itself includes compilation and prioritizes deterministic computations while delaying the truly non-deterministic parts [12]. This approach makes it possible, for instance, to realize a generalization of first-order unification in a completely deterministic fashion. The technique that is adopted for propagating types at run-time uses information already present during compilation to reduce substantially the effort to be expended dynamically [6]. Using this approach, virtually *no* new computation is required relative to a monomorphic subset of the language that is similar to Prolog. Finally, towards supporting modular programming, a method has been designed for realizing module interactions that permits separate compilation [14]. The actual addition and removal of modules of code can be achieved by methods developed for handling scoping over program clauses. However, these methods have been embellished by efficient devices for determining if a block of code will be redundant in a given context and also by mechanisms for realizing information hiding.

3. Realizing the Implementation Scheme. An implementation of λProlog on stock hardware based on the above ideas envisages four software subsystems: a compiler, a loader, an emulator for the abstract machine and a user interface. The function of the compiler is to process any given module of λProlog code, to certify its internal consistency and to ensure that it satisfies a promise determined by an associated signature and, finally, to translate it into a byte code form consisting of a 'header' part relevant to realizing module interactions and a 'body' containing sequences of instructions that can be run on the abstract machine. The purpose of the loader is to read in byte code files for modules that need to be used, to resolve names and absolute addresses using the information in the header parts of these files and to eventually produce a structure consisting of a block of code together with information for linking this code into a program context when needed. The emulator provides the capability of executing such code after it has been linked. Finally, the user interface allows for a flexibility in the compilation, loading and use of modules in an interactive session.

The Teyjus system embodies all the above components and comprises about 50,000 lines of C code. The functionality outlined above is realized in its entirety in a development environment. Also supported is the use of the compiler on the one hand and the loader and emulator on the other in standalone mode. The system architecture actually makes byte code files fully portable. Thus, λProlog modules can be distributed in byte code form, to be executed later using only the loader/emulator. Finally, the system includes a disassembler for the purpose of viewing the results of compilation.

4. Related Languages and Implementations. Logic and functional programming languages have been used often in the role of metalanguages. However, λProlog and Elf are, to our knowledge, the only languages that systematically support the higher-order abstract syntax approach. The language Qu-Prolog [4] incorporates a partial understanding of binding: in particular, it permits binding operators to be identified and recognizes equality modulo α-conversion but does not support unification relative to the full set of λ-conversion rules and does not also include primitives for recursing over binding structure.

The λProlog language has received several implementations in the past. All but one of these implementations have been in the form of interpreters, the most recent one being the Terzo interpreter [20] that is written in Standard ML. The interpreter based implementations have not profited from an in-depth analysis of the issues that are peculiar to realizing λProlog and, consequently, it is not reasonable to compare their design with that of the Teyjus system. The only other implementation to adopt a compilation approach is Prolog/MALI [2]. The core of this implementation is a memory management system called MALI that offers functionalities relevant to realizing logic programming languages. To a first approximation, λProlog programs are compiled to C programs that execute MALI commands for creating and interpreting goal structure. Certain inefficiencies, such as the copying of goal structure, appear to be inherent to this approach as opposed to the abstract machine based approach used in Teyjus. There are also differences in the treatment of specific aspects such as the representation and runtime manipulation of types, the representation of terms, the realization of reduction, and the implementation of the scoping primitives. Finally, Prolog/MALI appears to support a different notion of modularity. It is of interest to analyze the impact of all these differences and also to quantify their effect on performance, but a treatment of this issue is beyond the scope of this paper.

5. The System Presentation. We will demonstrate the Teyjus system and will expose the λProlog language as embodied in this implementation. We will also discuss examples that indicate the metalanguage capabilities of λProlog.

Acknowledgements. B. Jayaraman, K. Kwon and D.S. Wilson have assisted in the design of the λProlog abstract machine. L. Headley, S.-M. Perng and G. Tong have participated in the implementation effort. Support for this work has been provided by NSF at different stages under the grants CCR-8905825, CCR-9208465, CCR-9596119 and CCR-9803849.

References

1. B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliatre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin, A. Saïbi, and B. Werner. The Coq Proof Assistant Reference Manual – Version V6.1. Technical Report 0203, INRIA, August 1997.

2. Pascal Brisset and Olivier Ridoux. The compilation of λProlog and its execution with MALI. Publication Interne No 687, IRISA, Rennes, November 1992.
3. Amy Felty. Implementing tactics and tacticals in a higher-order logic programming language. *Journal of Automated Reasoning*, 11(1):43–81, August 1993.
4. Richard A. Hagen and Peter J. Robinson. Qu-Prolog 4.3 reference manual. Technical Report 99-03, Software Verification Research Centre, School of Information Technology, University of Queensland, 1999.
5. John Hannan and Dale Miller. From operational semantics to abstract machines. *Mathematical Structures in Computer Science*, 2(4):415–459, 1992.
6. Keehang Kwon, Gopalan Nadathur, and Debra Sue Wilson. Implementing polymorphic typing in a logic programming language. *Computer Languages*, 20(1):25–42, 1994.
7. Dale Miller and Gopalan Nadathur. A logic programming approach to manipulating formulas and programs. In Seif Haridi, editor, *IEEE Symposium on Logic Programming*, pages 379–388. IEEE Computer Society Press, September 1987.
8. Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
9. Gopalan Nadathur. A proof procedure for the logic of hereditary Harrop formulas. *Journal of Automated Reasoning*, 11(1):115–145, August 1993.
10. Gopalan Nadathur. An explicit substitution notation in a λProlog implementation. Technical Report TR-98-01, Department of Computer Science, University of Chicago, January 1998.
11. Gopalan Nadathur, Bharat Jayaraman, and Keehang Kwon. Scoping constructs in logic programming: Implementation problems and their solution. *Journal of Logic Programming*, 25(2):119–161, November 1995.
12. Gopalan Nadathur, Bharat Jayaraman, and Debra Sue Wilson. Implementation considerations for higher-order features in logic programming. Technical Report CS-1993-16, Department of Computer Science, Duke University, June 1993.
13. Gopalan Nadathur and Dale Miller. An overview of λProlog. In Kenneth A. Bowen and Robert A. Kowalski, editors, *Fifth International Logic Programming Conference*, pages 810–827. MIT Press, August 1988.
14. Gopalan Nadathur and Guanshan Tong. Realizing modularity in λProlog. Technical Report TR-97-07, Department of Computer Science, University of Chicago, August 1997. To appear in *Journal of Functional and Logic Programming.*
15. Gopalan Nadathur and Debra Sue Wilson. A notation for lambda terms: A generalization of environments. *Theoretical Computer Science*, 198(1-2):49–98, 1998.
16. Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.
17. Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon D. Plotkin, editors, *Logical Frameworks*. Cambridge University Press, 1991.
18. Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proceedings of the ACM-SIGPLAN Conference on Programming Language Design and Implementation*, pages 199–208. ACM Press, June 1988.
19. D.H.D. Warren. An abstract Prolog instruction set. Technical Note 309, SRI International, October 1983.
20. Philip Wickline and Dale Miller. The Terzo 1.1b implementation of λProlog. Distribution in NJ-SML source files. See `http://www.cse.psu.edu/~dale/lProlog/`, April 1997.

Vampire

Alexandre Riazanov and Andrei Voronkov

Computer Science Department, University of Manchester

1 General Description

Vampire is a resolution-based theorem prover for first-order classical logic. The current version implements ordered binary resolution with the set-of-support strategy and ordered hyperresolution. The competition version will have equality rules.

Vampire owes many of its design ideas to Otter [5]. It uses a saturation-based algorithm similar to that described in [3], but implemented more efficiently. Some planned (but not yet implemented) extensions of Vampire are directed toward the ideas underlying the design of SPASS [9].

2 Compiling and Indexing

The efficiency of Vampire is due to a combination of compilation and indexing used to implement all costly operations: resolution, hyperresolution, superposition, backward and forward subsumption, and some simplification rules. All indexes used in Vampire implement so-called “perfect filtering” [2,6]: they find exactly all elements in an index to which the operation is applicable. In addition, our algorithms for index-based retrieval compute the results of the operation on all these elements (for example, the most general unifiers when the operation is resolution), as proposed in [8].

We use code trees [8] for forward subsumption, path indexing [7] for backward subsumption and a variant of discrimination trees [4] for resolution. To give the reader a flavor of how we combine compilation and indexing, we describe in some detail how hyperresolution is implemented.

To implement hyperresolution efficiently in presence of indexing one has to solve the problem of simultaneous retrieval of unifiable terms. This problem can be formulated as follows: given terms $t_1, \ldots, t_n$ and indices $I_1, \ldots, I_n$, retrieve all possible tuples of terms $s_1, \ldots, s_n$ from the indices $I_1, \ldots, I_n$ respectively such that $\langle t_1, \ldots, t_n \rangle$ is unifiable with $\langle s_1, \ldots, s_n \rangle$, and for each such tuple return a most general unifier.

For solving the problem we use a very simple algorithm based on repetitive application of a retrieval operation *Retrieve* for *one term*: given an index I and a term t, $\mathit{Retrieve}(I, t)$ generates a sequence of all pairs s, σ such that the term s is in the index and σ is the most general unifier for s and t.

The algorithm *SimRetrieve* for *simultaneous* retrieval works as follows:

1. If the tuple of terms consists of one term t we simply apply the operation Retrieve to t and the corresponding index:
$$\mathit{SimRetrieve}(\langle I\rangle, \langle t\rangle) = \{\langle\langle s\rangle, \sigma\rangle \mid \langle\langle s\rangle, \sigma\rangle \in \mathit{Retrieve}(I, t)\}.$$
2. To compute $\mathit{SimRetrieve}(\langle I_1, \ldots, I_n\rangle, \langle t_1, \ldots, t_n\rangle)$, we first arbitrarily choose $t_i, i \in 1, \ldots, n$. Then we enumerate pairs from $\mathit{Retrieve}(\langle I_i, t_i\rangle)$ and for every such pair $\langle s, \sigma\rangle$ compute
$$\mathit{SimRetrieve}(\langle I_1, \ldots, I_{i-1}, I_{i+1}, \ldots, I_n\rangle, \langle t_1\sigma, \ldots, t_{i-1}\sigma, t_{i+1}\sigma, t_n\sigma\rangle).$$
$\mathit{SimRetrieve}(\langle I_1, \ldots, I_n\rangle, \langle t_1, \ldots, t_n\rangle)$ will consists of all pairs
$$\langle\sigma \circ \theta, \langle s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_n\rangle\rangle$$
such that
$$\langle\theta, \langle s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n\rangle\rangle \in \mathit{SimRetrieve}(\langle I_1, \ldots, I_{i-1}, I_{i+1}, \ldots, I_n\rangle, \langle t_1\sigma, \ldots, t_{i-1}\sigma, t_{i+1}\sigma, t_n\sigma\rangle).$$

To improve efficiency of simultaneous retrieval we use a version of *Retrieve* which exploits advantages of *compilation*. A call of *Retrieve* taking as the argument a term of the form $t\theta$, where t is one of the initial terms t_i and θ is a substitution collected at the previous steps of retrieval, uses a compiled version of the retrieval algorithm *specialized* for this term $t\theta$.

A heuristic is used to choose the next term t_i at Step 2 of the algorithm above. We choose the term which seems to have less unifiers in the index, to avoid retrieval of unifiers which will be rejected at the subsequent steps.

3 Other Features

Another interesting feature of Vampire is a strategy intended for the best performance when the time and/or memory are limited. We call it the *limited resource strategy*. This strategy is inspired by Otter's `max_memory` option, but unlike Otter's strategy with `max_memory` fixed in advance, we change the size limits on clauses based on periodical *runtime* evaluation of resources consumed so far.

The idea is as follows. When a time limit is given, Vampire makes estimations on what clauses (called *currently eligible clauses*) can be processed by the end of the time limit at all. For each newly generated clause it is checked whether it is currently eligible. If it is not, Vampire discards the clause. The estimation principle takes into consideration the current strategy and tries to calculate how the process of clause generation will develop by the time limit.

It is very difficult to define the right formulas for estimation. Inaccurate estimations can lead to two possible results:

1. *Too strong eligibility criteria.* Too many clauses are discarded by the strategy. In this case the prover terminates before the time limit expires because no new clauses can be generated.

2. *Too weak eligibility criteria.* Too few clauses are discarded. In this case when the prover terminates, the set of kept clauses contains many clauses to which no inference rules have been applied. Such clauses may have been kept for a long time, thus slowing down processing of new clauses (for example, because subsumption is applied to all kept clauses).

Our first implementations of the limited resource strategy often resulted in incorrect estimations of both kinds. For example, with the time limit of 300 seconds, Vampire sometimes terminated after 250 seconds because too many clauses have been discarded. The current version has better estimations: too strong eligibility criteria only result in loss of less than 1 second. Too weak eligibility criteria are more difficult to avoid: we still have cases when 40% of kept clauses were not used by the time limit. Correct estimations are especially difficult for clauses with many literals.

4 Future

The competition version will have equality rules: superposition and simplification. We are now making experiments on a non-standard splitting rule, different from the one used in SPASS, but we cannot yet report the results. Finally, we plan to implement some forms of constraints.

5 Performance

For the previous competitions it was always the case that several provers exposed better performance than the previous year winner. Therefore, it is difficult to predict the comparative performance of a prover based on the previous year results. However, newer versions of other provers are not yet released, so we can only present a comparison of Vampire with the previous year systems.

Since equality is not yet implemented in Vampire, we only show the results for problems without equality. The performance of other systems is taken from the Web page
`http://www.cs.jcu.edu.au/~tptp/CASC-15/CASC-15FullResults.html`.

We used a SPARC station with a performance similar to computers used at the competition. Our station has more memory, but we put the limit of 64M to simulate the conditions of CASC-15. We would like to note that Vampire's performance is not linear in the speed of a computer, because of the limited resource strategy.

Tables 1 and 2 show the performance of Vampire respectively on Horn and non-Horn problems as compared to the best systems of the 1998 competition CASC-15. Table 3 gives the summary for all 30 problems without equality.

For Horn problems, Vampires is able to solve all problems selected for the competition. For non-Horn problems, the only problem not solved by Vampire (`SYN067-3`) has not been solved by any other prover. In total, Vampire solves 29 problems out of 30, while SPASS that was the best on nonequality problems solved 24 problems.

	Gandalf c-1.1	SPASS 1.0.0a	p-SETHEO C-15	Otter 3.0.5	AI-SETHEO C-15	Vampire
LCL131-1	26.00	timeout	17.20	68.40	no_soln	33.15
LCL207-1	59.80	2.30	1.50	0.30	4.00	0.29
PLA014-1	35.50	0.30	2.50	timeout	5.90	0.06
LCL010-1	0.10	0.30	1.60	0.50	4.00	5.31
LCL211-1	0.30	13.40	1.40	4.70	4.90	0.70
PLA013-1	34.50	0.20	2.00	timeout	6.00	0.05
LCL115-1	1.70	18.90	no_soln	1.40	163.40	26.43
LCL093-1	5.10	timeout	no_soln	43.20	no_soln	38.26
LCL075-1	0.50	1.90	10.40	2.20	15.40	30.57
LCL018-1	timeout	timeout	no_soln	125.70	no_soln	37.43
LCL033-1	0.00	0.20	1.90	0.00	131.90	0.07
LCL129-1	timeout	262.90	no_soln	timeout	no_soln	247.10
SYN192-1	1.60	0.70	5.50	0.50	19.00	0.12
PLA018-1	37.20	1.00	2.60	timeout	5.10	0.42
LCL011-1	0.20	1.40	1.80	39.30	7.30	31.42
PLA012-1	38.70	2.60	3.30	timeout	5.50	0.71
LCL009-1	1.00	0.40	2.30	0.20	5.30	0.71
SYN310-1	0.00	0.20	197.80	0.00	113.50	1.25
LCL051-1	3.10	timeout	no_soln	7.90	no_soln	32.03
SYN190-1	2.10	1.40	5.70	0.50	22.80	0.23
Attempted	20	20	20	20	20	20
Solved	18	16	15	15	15	20
Av.Time	13.74	19.26	17.17	19.65	34.27	24.32

Vampire did not participate in the competition (and does not like light), so we put Vampire's column in gray shade. We also grayed the hard problems: those not solved by at least two of the best provers.

Table 1. Performance for Horn clause non-equality problems

	SPASS c-1.1	p-SETHEO 1.0.0a	AI-SETHEO C-15	Gandalf 3.0.5	Otter C-15	Vampire
SYN476-1	timeout	38.80	55.20	unknown	timeout	175.03
SYN480-1	129.00	38.00	62.80	unknown	timeout	120.16
CIV006-2	0.90	no_soln	no_soln	0.50	0.30	0.09
SYN510-1	149.80	25.10	61.40	unknown	timeout	190.94
PUZ005-1	0.50	2.50	14.40	0.40	0.10	0.07
SYN067-3	timeout	no_soln	no_soln	unknown	timeout	timeout
PUZ032-1	0.20	1.40	4.60	0.00	0.00	0.02
PUZ028-6	2.00	3.00	5.20	unknown	timeout	9.03
SYN036-1	2.40	no_soln	no_soln	106.60	timeout	0.23
SYN486-1	153.00	36.20	69.50	unknown	timeout	85.25
Attempted	10	10	10	10	10	10
Solved	8	7	7	4	3	9
Av.Time	54.73	20.71	39.01	26.88	0.13	58.08

Table 2. Performance for non-Horn non-equality problems

	SPASS c-1.1	Gandalf 1.0.0a	p-SETHEO C-15	AI-SETHEO 3.0.5	Otter C-15	Vampire
Attempted	30	30	30	30	30	30
Solved	24	22	22	22	18	29
Av.Time	31.08	16.13	18.66	35.78	16.40	36.80

Table 3. Overall performance for non-equality problems

Acknowledgments

In our previous life in Uppsala, we have been supported by TFR grants. We thank Christoph Weidenbach for his comments about the implementation of SPASS and Max-Planck Institut für Informatik for supporting a visit of the second author. We thank anonymous referees for their comments, among which our favorite is *"after all, Vampire is not a glass of Tuborg"*.

References

1. H. de Nivelle. An algorithm for the retrieval of unifiers from discrimination trees. *Journal of Automated Reasoning*, 20(1/2):5–25, January 1998.
2. P. Graf. *Term Indexing*, volume 1053 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.
3. E.L. Lusk. Controlling redundancy in large search spaces: Argonne-style theorem proving through the years. In A. Voronkov, editor, *Logic Programming and Automated Reasoning. International Conference LPAR'92.*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 96–106, St.Petersburg, Russia, July 1992.
4. William W. McCune. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(2):147–167, 1992.
5. W.W. McCune. OTTER 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, January 1994.
6. I.V. Ramakrishnan, R. Sekar, and A. Voronkov. Term indexing. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning.* Elsevier Science and MIT Press, 1999. To appear.
7. M. Stickel. The path indexing method for indexing terms. Technical Report 473, Artificial Intelligence Center, SRI International, Menlo Park, CA, October 1989.
8. A. Voronkov. The anatomy of Vampire: Implementing bottom-up procedures with code trees. *Journal of Automated Reasoning*, 15(2):237–265, 1995.
9. C. Weidenbach, B. Gaede, and G. Rock. Spass & flotter. version 0.42. In M.A. McRobbie and J.K. Slaney, editors, *Automated Deduction — CADE-13*, volume 1104 of *Lecture Notes in Computer Science*, pages 141–145, New Brunswick, NJ, USA, 1996.

System Abstract: E 0.3

Stephan Schulz

Institut für Informatik, Technische Universität München
D-80290 München, Germany
schulz@informatik.tu-muenchen.de

Abstract. We describe the main characteristics of version 0.3 of the E equational theorem prover. E is based on superposition and rewriting. It features a powerful interface for specifying search guiding heuristics. We discuss some important details of the implementation and demonstrate the performance of the prover by presenting experimental results on the TPTP. Finally, we describe our future plans for improving the system.

1 Introduction

E is a fully automatic theorem prover for clausal logic with equality. It is sound and, if used with fair strategies, complete.

We had two aims in designing and implementing this new prover. First, we were looking for a new platform to continue our work on proof analysis and learning. While DISCOUNT [DKS97], the system used for our previous work, has been surprisingly adaptable, the code base was showing its age, and each new extension became more and more difficult. Moreover, DISCOUNT is restricted to unit equational proof problems, while most current application problems are more naturally encoded in full clausal logic.

Secondly, we wanted to improve the relatively poor performance of SETHEO [MIL$^+$97] on problems with equality. The METOP calculus [Mos96] was designed to allow efficient handling of equality in a model-elimination prover. It requires a saturation phase to be interleaved with the top-down proof search.

Thus, E was designed as an efficient and flexible inference engine that enables us to easily implement the necessary features. In this paper we concentrate on the use of E as a stand-alone prover and only briefly mention special features resulting from the intended use of E within a METOP-based prover.

2 Calculus

E is a purely equational theorem prover, based on ordered paramodulation and rewriting. As such, it implements an instance of the superposition calculus described in [BG94]. We give a very short introduction to the calculus, discussing special aspects of E.

$Term(F, V)$ denotes the set of (first order) *terms* over a set F of function symbols and set V of variables. We write $t|_p$ to denote the subterm of t at a position p and write $t[p \leftarrow t']$ to denote t with $t|_p$ replaced by t'. An equation

$s = t$ is an (implicitly symmetrical) pair of terms. A positive literal is an equation $s = t$, a negative literal is a negated equation $s \neq t$. Literals can be represented as multi-sets of multi-sets of terms, with $s = t$ represented as $\{\{s\}, \{t\}\}$ and $s \neq t$ represented as $\{\{s, t\}\}$. A *ground reduction ordering* $>$ is a Noetherian partial ordering that is stable w.r.t. the term structure and substitutions and total on ground terms. $>$ can be extended to an ordering $>_1$ on literals by comparing the multi-set representation of literals with $\ggg\ggg$ (the multi-set-multi-set extension of $>$). Alternatively, we define $>_2$ on literals as $>_1 \setminus (\{(s = t, s' \neq t') | s, t, s', t' \in \mathit{Term}(F, V)\} \cup \{(s \neq t, s' = t') | s, t, s', t' \in \mathit{Term}(F, V)\})$, i.e. by making positive and negative literals incomparable.

By default, E implements the generating inference rules *Equality Resolution*, *Superposition (left/right)* and *Equality Factoring* as described in [BG94], i.e. restricted to maximal terms (w.r.t. $>$) and maximal literals (w.r.t. $>_1$).

From the many possibilities of simplifying or eliminating clauses in the superposition calculus, E implements *unconditional rewriting*, *subsumption*, *deletion of tautologies*, and a new contracting inference rule named *Simplify-Reflect* that allows us to directly eliminate negative literals that are subsumed by positive unit clauses, but cannot be rewritten:

$$\text{(SR)}\quad \frac{\mathtt{s} = \mathtt{t} \qquad \mathtt{u}[\mathtt{p} \leftarrow \sigma(\mathtt{s})] \neq \mathtt{u}[\mathtt{p} \leftarrow \sigma(\mathtt{t})] \vee \mathtt{R}}{\mathtt{s} = \mathtt{t} \qquad \mathtt{R}} \qquad \begin{array}{l}\text{if } \sigma(s), \sigma(t) \text{ are} \\ >\text{-incomparable}\end{array}$$

As (SR) only removes clauses that are *composite* with respect to the remaining set of clauses, the calculus is complete. For the case of unit clauses, it degenerates into *unfailing completion* [BDP89] as implemented in DISCOUNT.

In addition to the above general saturation strategy, E also implements the positive unit strategy for Horn clauses described in [Der91]. This is achieved by substituting $>_1$ with $>_2$ in the generating inference rules and by restricting the superposition rule to paramodulate only from positive unit clauses and into negative literals and positive unit clauses. As a third option, an intermediate strategy (paramodulating only from positive unit clauses, but into arbitrary literals) is available, and has been quite successful in the experimental evaluation.

Contrary to e.g. SPASS [WGR96], E does not implement special rules for non-equational literals or sort theories, as we expect this part to be taken care of by SETHEO in a later combined system. Instead, non-equation literals are encoded as equations and dealt with accordingly.

3 Search Control

The basic proof procedure of E is straightforward: The set of all clauses is split into a set of *processed* clauses and a set of *unprocessed* clauses. Initially, all input clauses are unprocessed, and the set of processed clauses is empty. The prover selects an unprocessed clause, simplifies it w.r.t. to the processed clauses, then uses it to simplify the processed clauses in turn. It then performs equality factoring, equality resolution and superposition between the selected clause and the set of processed clauses. The generated clauses are added to the set of unprocessed clauses. Processed clauses in which maximal terms have been subject to simplification inferences are also moved to the set of unprocessed clauses for reconsideration. The process stops when the empty clause is derived or no further inferences are possible.

There are two main parameters to the proof procedure: The reduction ordering and the order in which unprocessed clauses are selected. E currently implements KBO and LPO, with a variety of simple schemes to generate precedences and weights. A simple default is used unless otherwise specified.

The order in which unprocessed clauses are selected is determined by a search control heuristic. Such a heuristic sets up a variety of priority queues and a weighted round-robin scheme that determines from which queue the next clause is to be selected. Order within each queue is determined, in this order, by a *priority function* that can e.g. prefer all-negative clauses, ground clauses, or initial axioms, and by an *evaluation function* that is typically based on symbol counting. Completeness of the prover can be guaranteed by careful selection of priority and evaluation functions, or by simple addition of a fair queue (e.g. a FIFO-queue). Search control heuristics can be chosen from a predefined set or completely specified at run time. The prover also supports a simple automatic mode in which a heuristic is chosen based on some simple properties of the problem.

4 Implementation

E is build on top a layer of libraries that offer successively more specialized services, from low level I/O to the main proof search algorithm. The most outstanding feature probably is the use of perfectly shared terms. Except for short-lived temporary copies, each distinct subterm is represented exactly once in E. This means that each individual rewriting step potentially affects a large number of terms, literals and clauses. As an example, a single application of the equation $f(x) = x$ to any subterm of the form $f(a)$ simultaneously transforms $g(f(a), f(f(f(a)))$ into $g(a, a)$. This is achieved by recursively propagating the replacing of a term to all its superterms while ensuring that multiple identical copies created by this replacing collapse onto a single instance. Term positions on which rewriting is restricted by the calculus are kept distinct from normal terms and are thus exempt from this behaviour. Sharing of identical subterms reduces the number of allocated term cells by at least one order of magnitude, and allows very efficient detection of terms already in normal form with respect to a given set of rewrite rules.

The second important feature of the implementation is the use of *perfect discrimination trees* [Gra95] for rewriting, unit-subsumption and *Simplify-Reflect*. The performance of the discrimination trees is further improved by the use of age and size constraints on the branches of the tree. This allows us to backtrack from branches that cannot lead to a successful match very early.

E is implemented in ANSI-C. It has been successfully compiled and installed under various versions of GNU/Linux, SunOS, Solaris and HP-UX, using both the GNU C compiler and SUN's proprietary C compiler.

5 Experimental Results

As a result of the powerful way in which search control heuristics can be specified in E, the space of possible parameters is even bigger than for many other fully automatic theorem provers. We will therefore only present results of E in *fully automatic* mode, i.e. in the way a naive user would use the system. For comparison, we include data for SPASS 0.85 and Otter [MW97] on comparable hardware and with similar time limits[1].

We used the set of all clause normal form problems from the TPTP [SSY94] problem library, version 2.1.0, for evaluation. Results are presented for the complete set and for subsets described by the structure of the clauses and the equality content of the problem.

	No Equality			Equality			Overall
	Unit	Horn	Non-Horn	Unit	Horn	Non-Horn	
Problems	11	557	766	402	321	1218	3275
	Proofs						
E 0.31a	8	461	286	312	252	301	1620
Otter/Mace	8	448	294	289	207	276	1522
SPASS 0.85	8	434	349	261	201	350	1603
	Models						
E 0.31a	3	5	48	1	3	3	63
Otter/Mace	3	21	97	10	8	3	142
SPASS 0.85	3	5	110	1	0	3	122
	Total successes						
E 0.31a	11	466	334	313	255	304	1683
Otter/Mace	11	469	391	299	215	279	1664
SPASS 0.85	11	439	459	262	201	353	1725
	Average time (seconds/solution)						
E 0.31a	0.03	12.63	18.60	15.06	29.34	22.45	18.49
Otter/Mace	0.73	8.26	6.52	3.01	6.71	13.33	7.51
SPASS 0.85	0.01	8.34	20.29	14.26	26.57	15.25	15.90

Results for E have been obtained with the latest version, 0.31a, and in compliance with the guidelines for use of the TPTP. TPTP input files were unchanged except for removal of equality axioms and syntax transformation. The performance was evaluated on a cluster of SUN Ultra 60 workstations running at 300 MHz. CPU time per attempt was limited to 300 seconds, memory to 192 MB. Results for SPASS have been obtained on Pentium-II computers running at 300 MHz, and with a time limit of 300 seconds. Otter results are for the combination of Otter 3.0.5 and Mace 1.3.2 that participated in the CASC-15 ATP competition, running on Pentium-II computers at 400 MHz, and with a cut-off time of 300 seconds (as opposed to the 600 seconds on the Otter web page). In our experience, a 400 MHz Pentium-II computer performs within 5% of our SUN workstations for measuring CPU times on theorem proving tasks.

As the table shows, E finds slightly more proofs than either of the other two provers over the complete TPTP. It performs particularly well for unit problems,

[1] We thank Bill McCune, Geoff Sutcliffe and Christoph Weidenbach for their assistance with obtaining and interpreting the data.

which are mostly pure equational, and Horn problems, where the three different saturation strategies complement each other well. E is slightly less successful for general clauses, especially in the non-equational case. We assume that the main reason for this is the lack of specialized inference rules for non-equational literals, which play an important role most general problems even if equality is present. Finally, E is not very powerful for showing satisfiability. This is not surprising, as we have made no attempt to improve the calculus in this direction.

6 Future Work

We believe that the results in the previous section show the potential of our implementation. However, a lot of work remains to be done. In particular, we want to create tools for proof presentation and analysis. We also will improve on the selection of evaluation functions, including goal-oriented and learning heuristics analogous to those implemented in DISCOUNT, and will automate the process of generating and/or selecting suitable term orderings.

Finally, we plan to integrate E and SETHEO to form a METOP-based hybrid system combining the strengths of the top-down and the bottom-up approach.

References

BDP89. L. Bachmair, N. Dershowitz, and D.A. Plaisted. Completion Without Failure. *Coll. on the Resolution of Equations in Algebraic Structures, Austin, 1987.* Academic Press, 1989.

BG94. L. Bachmair and H. Ganzinger. Rewrite-Based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 3(4):217–247, 1994.

Der91. N. Dershowitz. Ordering-Based Strategies for Horn Clauses. *12th IJCAI, Sydney*, pp. 118–124. Morgan-Kaufmann, 1991

DKS97. J. Denzinger, M. Kronenburg, and S. Schulz. DISCOUNT: A Distributed and Learning Equational Prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.

Gra95. P. Graf. *Term Indexing.* LNAI 1053. Springer, 1995.

MIL$^+$97. M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. SETHEO and E-SETHEO – The CADE-13 Systems. *Journal of Automated Reasoning*, 18(2):237–246, 1997.

Mos96. M. Moser. *Goal-Directed Reasoning in Clausal Logic with Equality.* CS Press, München, 1996. Ph.D. Thesis, Fakultät für Informatik, TU München.

MW97. W.W. McCune and L. Wos. Otter: The CADE-13 Competition Incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997.

SSY94. G. Sutcliffe, C.B. Suttner, and T. Yemenis. The TPTP Problem Library. *Proc. CADE-12, Nancy*, LNAI 814, pp. 252–266. Springer, 1994.

WGR96. C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER Version 0.42. *Proc. CADE-13, New Brunswick*, LNAI 1104, pp. 141–145. Springer, 1996.

The E 0.3 distribution is available on the WWW at
`http://wwwjessen.informatik.tu-muenchen.de/~schulz/WORK/eprover.html`

Rewrite-Based Deduction and Symbolic Constraints

Robert Nieuwenhuis*

Technical University of Catalonia, Dept. LSI
Jordi Girona 1, 08034 Barcelona, Spain
roberto@lsi.upc.es

1 Introduction

Building a state-of-the-art theorem prover requires the combination of at least three main ingredients: good theory, clever heuristics, and the necessary engineering skills to implement it all in an efficient way. Progress in each of these ingredients interacts in different ways.

On the one hand, new theoretical insights replace heuristics by more precise and effective techniques. For example, the completeness proof of *basic paramodulation* [NR95,BGLS95] shows why no inferences below Skolem functions are needed, as conjectured by McCune in [McC90]. Regarding implementation techniques, ad-hoc algorithms for procedures like demodulation or subsumption are replaced by efficient, re-usable, general-purpose indexing data structures for which the time and space requirements are well-known.

But, on the other hand, theory also advances 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 considered a "key strategy" by McCune for finding his well-known proof of the Robbins conjecture [McC97] by basic paramodulation modulo associativity and commutativity (AC).

Provers like Spass [Wei97], 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 (as we will see later on).

One obstacle for progress in the development of such provers seems to be the scarcity of large enough research teams with knowledge in the three aspects: theory, heuristics and implementation techniques. (Another problem may be that implementation efforts seem to produce less publications on a researcher's CV than the same efforts on theory.)

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 (first-order clausal logic). In our opinion, deduction

* Partially supported by the ESPRIT Basic Research WG CCL-II.

with constraints is an adequate paradigm for doing this in a clean way. It uses specialized (constraint solving) 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. For instance, in paramodulation- or resolution-based systems, a constrained clause $C \mid T$ represents the set of ground instances of the clause part C that satisfy the constraint T, and unification is replaced by equality constraint solving.

Here we will address the current theoretical and practical challenges concerning the construction of saturation-based provers by ordered paramodulation techniques and symbolic constraints. We will briefly survey the considerable amount of recent progress that has been made on the theoretical side, and mention some of the most promising ideas that are ready for use in actual provers. At the same time, we will describe a number of open problems (of both theoretical and practical nature) that still need to be solved in order to make other theoretical advances applicable. In some cases we will point to possible solutions for these problems.

A special attention will be devoted to *redundancy proving* and its very close relationship with *implicit* inductive theorem proving, that is, where the user does not need to provide explicit induction schemes.

2 Ordered Paramodulation, Redundancy, and Saturation

We now very shortly survey some fundamental theoretical results in this area. For more details and further references on most of the contents of this section, see [BG98,NR99b].

The inference rule for paramodulation[1] [RW69]

$$\frac{C \vee s \simeq t \qquad D}{(C \vee D[t]_p)\sigma} \qquad \text{where } \sigma = mgu(s, D|_p)$$

has been further restricted in many ways. For example, its refutation completeness is preserved when applied only if $D|_p$ is not a variable, and if, for some ordering $\succ$ on terms and equations, the paramodulation steps involve only maximal terms of maximal equations of both premises. In this case it is called *superposition* [BG94]. *Ordered paramodulation* is the slightly less restricted version of superposition where inferences also take place *on* non-maximal sides of equations [HR91].

[1] Here $D|_p$ denotes the subterm of the clause D at position p, and $D[t]_p$ denotes the result of replacing in D that subterm by t. If $D|_p$ is in the term u of a (positive or negative) equation $u \simeq v$ in D, then we say that the paramodulation step *involves* the terms s and u; more precisely, it takes place *with* s *on* u.

2.1 Paramodulation with Constrained Clauses

By expressing the ordering and unification restrictions as inherited constraints, paramodulation remains complete [NR95]. Then it becomes:

$$\frac{C \vee s \simeq t \mid T \qquad D \mid T'}{C \vee D[t]_p \mid T \wedge T' \wedge s = D|_p \wedge OC}$$

where the part OC of the constraint represents the ordering restrictions, that is, OC is of the form $s > t \wedge \ldots$ Here the symbols $=$ and $>$ in the constraints are interpreted, respectively, as syntactic equality $\equiv$ of terms and as the given ordering $\succ$ on terms. A main advantage 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 on the current inference rule *and on the previous ones*), then it produces a conclusion with an unsatisfiable constraint, i.e., a tautology, and hence the inference is not needed. The basicness restriction (no inferences computed on terms coming from unifiers of ancestor inferences) [BGLS95,NR95] in this notation is a natural consequence of the fact that inferences only take place on the clause part C of $C \mid T$, and no unifiers are ever applied to C.

2.2 Paramodulation Modulo *E*

It is well-known that paramodulation with some axioms, like the AC axioms, generates many slightly different permuted versions of clauses. Hence for efficiency reasons it is many times better to treat all these clauses together as a single one representing the whole class. This leads to the notion of building-in equational theories E, where the axioms E are removed, and instead one uses E-paramodulation:

$$\frac{C \vee s \simeq t \qquad D}{(C \vee D[t]_p)\sigma} \qquad \text{for all } \sigma \in mgu_E(s, D|_p)$$

where $mgu_E(s,t)$ denotes a complete set of E-unifiers of s and t. For example, for the case where E consists of the AC-axioms for some function symbols, a procedure was first given in [PS81]. If f is such an AC-symbol, then by (purely equational) E-paramodulation with $f(a,x) \simeq x$ on $f(b,y) \simeq g(y)$ we can infer $b \simeq g(a)$, and also $f(b,z) \simeq g(f(a,z))$, where z is a new variable.

2.3 E-Paramodulation with Constraints

When dealing with constrained clauses, E-paramodulation is expressed exactly as ordinary paramodulation. *The only aspect that changes is the interpretation of the constraints:* now $=$ is interpreted as E-equality, instead of syntactic equality[2]. Apart from the basicness restriction, an additional advantage is now that only one conclusion is generated, instead of one conclusion for each E-unifier[Vig94,NR97]. This can have dramatic consequences. For example, there are more than a million unifiers in $mgu_{AC}(f(x,x,x), f(y_1,y_2,y_3,y_4))$.

[2] Although for some E *extended* inference rules are needed (see, e.g., [RV95]).

2.4 Redundancy and Saturation

Roughly, a clause C is *redundant* in a set of clauses S if C is a logical consequence of smaller (with respect to the given clause ordering) clauses of S. This abstract notion covers well-known practical simplification and elimination techniques like demodulation or subsumption, as well as many other more powerful methods. A similar notion of redundancy of inferences exists as well, and a set of clauses is called *saturated* for a given inference system $\mathcal{I}$ if it is closed under $\mathcal{I}$, up to redundant inferences. Roughly, *saturation* is a procedure that adds conclusions of non-redundant inferences and removes redundant clauses. In the limit such a procedure produces a saturated set.

Saturation with respect to appropriate $\mathcal{I}$ is refutation complete: a model can be built for every saturated set not containing the empty clause [BG94]. This is not only of theoretical value: Spass successfully applies finite saturation to prove satisfiability for non-trivial problems (winning in the corresponding category of the last two CADE ATP System Competitions).

The availability of decision procedures for subproblems is well-known to be very useful in a prover. The ideas explained so far in this section have also generated a number of new results in this direction. For example, superposition with simplification can be used as a decision procedure for the monadic class with equality [BGW93b] (which is equivalent to a class of set constraints [BGW93a]). Similar very recent results have been obtained for the guarded fragment [GMV99,GdN99]. Results on the complexity and decidability of unification problems in Horn theories have been obtained by basic paramodulation [Nie98] and sorted superposition [JMW98].

3 Perspectives

We now mention some of the most promising ideas in this field that are ready for use in actual provers, and describe a number of open problems (of both theoretical and practical nature) that still need to be solved in order to make other theoretical advances applicable.

3.1 Basicness and Redundancy

From several practical experiments it is known that the basic restriction in paramodulation (modulo the empty theory) saves a large amount of (in some sense, repeated) work and behaves quite well in practice. It is also easy to implement in most provers by marking *blocked* subterms, i.e., the point where the constraint starts.

Open Problem 1: However, it is well-known that full simplification by demodulation is incomplete in combination with the basic strategy (see [NR95] for counter examples). Although some ideas are given in [LS98], better results are needed for practice. We conjecture (this conjecture is supported by our practical experiments) that unrestricted *forward* demodulation (and, in general, forward redundancy), which is where 90% of the time is spent in most provers, does not lead to incompleteness. For backward demodulation, *weakening* of the constraint (as explained in [NR95]) would still be needed.

Let us now look at simplification in the context of E-paramodulation with constraints. Consider again McCune's AC-paramodulation proof of the Robbins conjecture, where no constraints were used. Instead, for each paramodulation inference, complete sets of unifiers were still computed, and one new equation added for each one them (although heuristics were used to discard some of the unifiers). One possible reason for not using constraints might have been the next open problem:

Open Problem 2: How can we apply a constrained equation $s \simeq t \mid T$ in a demodulation step without solving the E-unification problem in T? A quite naive solution could be the following. 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.

3.2 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.

Open Problem 3: 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. Furthermore, until very recently, all completeness results for ordered paramodulation required the term ordering $\succ$ to be well-founded, monotonic and total(izable) on ground terms. However, in E-paramodulation, the existence of such a total *E-compatible* ordering is a very strong requirement. For example, a large amount of work has been done on the development of progressively more suitable AC-compatible orderings. But for many E such orderings cannot exist at all. This happens for instance when E contains an idempotency axiom.

Open Problem 4: Very recently, the monotonicity requirement on the ordering has been dropped for ordered paramodulation [BGNR99]. However, many open questions remain concerning the completeness of full superposition and the compatibility with redundancy notions when working with non-monotonic orderings (see [BGNR99] for details).

Open Problem 5: Also, more research is needed for developing suitable E-compatible orderings (monotonic as well as non-monotonic ones), and for studying their practical behaviour on different problem domains. For example, a nice and simple non-monotonic AC-compatible ordering can be obtained by using RPO on *flattened* terms. What can be done for other theories E?

3.3 Constraint Solving

When working modulo the empty theory, equality constraint solving is just syntactic unification.

Regarding ordering constraint solving, many algorithms of a theoretical nature have been given for path orderings, starting with [Com90]. Although deciding the satisfiability of path ordering constraints is NP-complete for all relevant cases, very recently, a family of path ordering constraint solving algorithms of a more practical nature has been given [NR99a].

Open Problem 6: Is there any useful ordering for which deciding the satisfiability of (e.g., only conjunctive) constraints is in P? (Although for size-based orderings the problem looks simple, it seems that one quickly runs into linear diophantine (in)equations...) Or, even if not in P, for which orderings can we have good practical algorithms?

Open Problem 7: 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?

Open Problem 8: 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 [BS99]). And, as in the previous open problem, are there any sound tests detecting most cases of unsatisfiability? Note that, when the empty clause with a constraint T is found, this denotes an inconsistency only if (the equality part of) T is satisfiable. Hence, unlike what happens with ordering constraints, at least then it is necessary to really prove the satisfiability (and not the unsatisfiability) of constraints. But this still does not require a decision procedure; a semi-decision procedure suffices, and always exists (e.g., by narrowing). This may be useful when the decision problem is extremely hard (associativity) or undecidable (associativity $\cup$ distributivity and extensions).

Open Problem 9: Once the adequate orderings for E-paramodulation have been found (see open problem 5), is there any reasonably efficient ordering constraint solving procedure for them?

3.4 Indexing Data Structures

As said, for many standard operations like many-to-one matching or unification indexing data structures have been developed that can be used in operations like inference computation, demodulation or subsumption. Such data structures are crucial in order to obtain a prover whose throughput remains stable while the number of clauses increases.

Open Problem 9: For many operations no indexing data structures have been developed yet. For example, assume we use demodulation with unorientable equations, and such an equation $s \simeq t$ is found to be applicable to a term $s\sigma$. Then after matching we have to check 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.

Open Problem 10: 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?

3.5 More Powerful Redundancy Notions

In the *Saturate* system [GNN95], a number of experiments with powerful redundancy notions has been carried out. For example, *constrained rewriting* turns out to be powerful enough for deciding the confluence of ordered rewrite systems [CNNR98]. 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 our experiments it is clear that such techniques importantly reduce the number of retained clauses.

Open Problem 11: 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.

3.6 Redundancy and Inductive Theorem Proving

Let us now consider the close relationship between redundancy and inductive proofs[3]. More precisely, we aim at (semi-)automatically proving or disproving inductive validity: given a set of Horn clauses H (the axioms), a clause C (the conjecture) is inductively valid if it is valid in the minimal (or initial) Herbrand model I of H.

Example 1. Let H consist of the 3-axiom group presentation

$$\{ \quad e + x = x, \quad i(x) + x = e, \quad (x + y) + z = x + (y + z) \quad \}$$

over a signature $\mathcal{F}$ is $\{e^0, a^0, i^1, +^2\}$ (arities are written as superscripts). Then I (a group with one generator a) is isomorphic to the integers with $+$ and if the

[3] This relationship was already noted in [GS92] and used in a different setting.

conjecture C is $\forall x, y.\ x + y = y + x$ then $I \models C$. Note that this is not the case if there is another generator b, since then $a + b = b + a$ does not hold.

In order to minimise user interaction and hence the amount of needed user expertise, we do not want to require the user to provide *explicit* induction schemes. Instead, we use what has been called *implicit* induction[4], based on well-founded general-purpose term orderings $\succ$.

Our ideas are based on a well-known naive method: enumerate the set of all ground instances of the conjecture C, by progressively instantiating it.

Example 2. Let us continue with the previous example and let C be the (false) conjecture $(x + x) + y = y$. Then we can choose an arbitrary variable of C, say x, as the *induction variable,* and instantiate it in all possible ways, that is, with all members of the set $\{f(x_1 \ldots x_n) \mid f^n \in \mathcal{F}\}$ where the x_i are distinct *fresh* variables. By such an *expansion* step of the induction variable, we obtain four new conjectures, namely $(e{+}e){+}y = y$, $(a{+}a){+}y = y$, $(i(z){+}i(z)){+}y = y$, and $((z{+}w){+}(z{+}w)){+}y = y$ (note that, independently of the chosen variable, every ground instance of C is also an instance of one of the four new conjectures). If we continue with $(a + a) + y = y$, in one more step the ground counter example $(a + a) + e = e$ (among others) is obtained, thus disproving $I \models C$.

Indeed, under the assumption that $I \models D$ is decidable for ground clauses D, the method is refutation complete: if $vars(C) = \{x_1 \ldots x_n\}$ then any ground counter example $C\sigma$ can be obtained by $|x_1\sigma| + \ldots + |x_n\sigma|$ expansion steps. Hence it will eventually be found if expansion is applied with fairness. Note that if the validity of ground conjectures is undecidable, refutation completeness is impossible anyway. But of course validity (dis)proofs can then still be obtained in many cases.

Example 3. Let $\mathcal{F}$ be $\{0^0, s^1, +^2\}$ and assume H consists of the equations $0{+}x = x$ and $s(x){+}y = s(x{+}y)$. Then I is isomorphic to the algebra of natural numbers with $+$. In the case of a constructor discipline, expansion can be limited to constructor instances (here, with 0 and $s(y)$). Therefore, the (false) conjecture $x{+}x = x$ can be expanded in two ways, into $0{+}0 = 0$ and $s(y){+}s(y) = s(y)$. The instance $0 + 0 = 0$ is valid in I, and hence we continue with $s(y) + s(y) = s(y)$, getting $s(0){+}s(0) = s(0)$ and $s(s(z)){+}s(s(z)) = s(s(z))$. Since $s(0){+}s(0) = s(0)$ is a counterexample, the conjecture is disproved. This process can be seen as a tree:

```
           x+x=x
         /       \
     0+0=0    s(y)+s(y)=s(y)
                /              \
       s(0)+s(0)=s(0)     s(s(z))+s(s(z))=s(s(z))
```

[4] But our method does not fall under the *proof-by-consistency* or *inductionless induction* techniques either.

Now let us consider validity proving. Since the leaves of the tree cover all possible instances of the conjecture at the root, it is clear that the root is proved valid if at a certain point all leaves can be shown valid.

Our main point here is that *any standard redundancy proving method can be used for proving the validity of such leaves C*, since it simply amounts to showing that $H \cup L \cup S_{<C\sigma} \models C\sigma$ for all ground instances $C\sigma$ of C, where L is a set of *lemmas*, arbitrary first-order clauses that are already known to be valid in I, and $S_{<C\sigma}$ is the set of nodes of the tree whose complexity measure (wrt. $\succ$) is smaller than the one of $C\sigma$, and can hence be assumed to be valid in I by induction hypothesis.

Note that lemmas can come from user input (proved valid by any induction method) or can be leaves that have already been proved valid, and that both lemmas and axioms can be applied without any ordering restrictions. We also remark that valid ground leaves are always provable only from H.

Redundancy is not only needed in the final stage for proving validity of all leaves. Since nodes that are known to be valid require no further expansion, it is also crucial for efficiency to apply it *during* validity (dis)proofs.

Example 4. Continuing with the previous example, $\forall x, y.\ \ x + y = y + x$ is an inductive theorem. The tree is as follows:

```
                        x+y=y+x
                      /         \
             0+y=y+0                s(z)+y=y+s(z)
           /        |                  |           \
0+0=0+0    0+s(z)=s(z)+0   s(z)+0=0+s(z)  s(z)+s(w)=s(w)+s(z)
  (1)           (2)              (3)                   (4)
```

All leaves are redundant, and hence the theorem is proved. For instance, for leaves (2) and (3) the following redundancy proof applies: $s(z) + 0$ is equivalent to $s(z + 0)$ by the second axiom, which by applying a smaller instance of the root node becomes $s(0 + z)$ and then, by the first axiom, becomes $s(z)$, which by the first axiom is equivalent to $0 + s(z)$.

Note that, unlike what happens in other implicit induction methods, these proofs are quite readible: they simply consist of case analysis and redundancy proofs. One can of course deal with a forest instead of a single tree in case several conjectures are given as input. This will be useful for *simultaneous induction.* Furthermore, unlike other implicit induction methods, here any well-founded ordering can be used, and there are no requirements on H or C (like saturatedness). We also remark that the method is not only applicable in the case of initial semantics, but for any semantics based on Herbrand models (like the class of *all* such models, or, when H is non-Horn, for *perfect model* semantics [GS92]).

Open Problem 12: There are many degrees of freedom in the schematic view of the induction method given here (nodes can be simplified, different notions of complexity of a node can be used, other kinds of expansion, etc.). How can we find the right settings in practice? In principle, all known redundancy methods that fit into the abstract notion of redundancy can be used. What kind of concrete methods will be useful?

Open Problem 13: Of course many well-known difficulties in inductive theorem proving appear as well here. For example, the conjecture may have to be generalized (e.g., in the previous example, $(x+x)+x = x+(x+x)$ cannot be proved directly, but associativity of $+$ is an easily provable generalization of it). Lemmas (other than generalisations of the conjecture) may be needed as well. How to find the right lemmas and generalisations? In particular, how can they be found from failed redundancy proofs?

Open Problem 14: It seems that superposition on $H \cup \{C\}$ sometimes finds good lemmas. This explains some of the successes of proof-by consistency methods. But we believe that the main drawback of the latter method is that it actually tries to prove *all* these superposition consequences. Can a hybrid method be useful, applying and proving by expansion only the useful lemmas obtained by superposition?

4 Concluding Remark

We hope that the reader has become motivated for building provers that take profit of the recent theoretical results that are now available, and challenged by some of the open problems that have been described.

References

BG94. Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

BG98. Leo Bachmair and Harald Ganzinger. Equational reasoning in saturation-based theorem proving. In W. Bibel and P. Schmitt, editors, *Automated Deduction: A Basis for Applications.* Kluwer, 1998.

BGLS95. L. Bachmair, H. Ganzinger, Chr. Lynch, and W. Snyder. Basic paramodulation. *Information and Computation*, 121(2):172–192, 1995.

BGNR99. Miquel Bofill, Guillem Godoy, Robert Nieuwenhuis, and Albert Rubio. Paramodulation with non-monotonic orderings. In *14th IEEE Symposium on Logic in Computer Science (LICS)*, Trento, Italy, July 2–5, 1999.

BGW93a. Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Set constraints are the monadic class. In *Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 75–83, Montreal, canada, June 19–23, 1993. IEEE Computer Society Press.

BGW93b. Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. 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, pages 83–96. SpringerVerlag, 1993.

BS99. Franz Baader and Wayne Snyder. Unification theory. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning.* Elsevier Science Publishers (to appear), 1999.

CNNR98. Hubert Comon, Paliath Narendran, Robert Nieuwenhuis, and Michael Rusinowitch. Decision problems in ordered rewriting. In *13th IEEE Symposium on Logic in Computer Science (LICS)*, pages 410–422, Indianapolis, USA, June 27–30, 1998.

Com90. Hubert Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.

GdN99. Harald Ganzinger and Hans de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *14th IEEE Symposium on Logic in Computer Science (LICS)*, Trento, Italy, July 2–5, 1999.

GMV99. Harald Ganzinger, Christoph Meyer, and Margus Veanes. The two-variable guarded fragment with transitive relations. In *14th IEEE Symposium on Logic in Computer Science (LICS)*, Trento, Italy, July 2–5, 1999.

GNN95. Harald Ganzinger, Robert Nieuwenhuis, and Pilar Nivela. The Saturate System, 1995. Software and documentation available at: `http://www.mpi-sb.mpg.de/SATURATE/Saturate.html`.

GS92. Harald Ganzinger and Jürgen Stuber. Inductive theorem proving by consistency for first-order clauses (extended abstract). In M[ichaël] Rusinowitch and J[ean-]L[uc] Rémy, editors, *The Third International Workshop on Conditional Term Rewriting Systems, Extended Abstracts*, pages 130–135, Pont-à-Mousson, France, July 8–10, 1992. Centre de Recherche en Informatique de Nancy and INRIA Lorraine.

HR91. J. Hsiang and M Rusinowitch. Proving refutational completeness of theorem proving strategies: the transfinite semantic tree method. *Journal of the ACM*, 38(3):559–587, jul 1991.

JMW98. Florent Jacquemard, Christoph Meyer, and Christoph Weidenbach. Unification in extensions of shallow equational theories. In *Proceedings of the 9th International Conference on Rewriting Techniques and Applications, RTA-9*, volume to appear, Tsukuba, Japan, 1998. Springer.

LS98. C. Lynch and C. Scharff. Basic completion with E-cycle simplification. In *Artificial Intelligence and Symbolic Computation*, lncs 1476, pages 121–121, 1998.

McC90. William McCune. Skolem functions and equality in automated deduction. In Tom Dietterich and William Swartout, editors, *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 246–251, Hynes Convention Centre?, July 29–August 3 1990. MIT Press.

McC97. William McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, December 1997.

Nie98. Robert Nieuwenhuis. Decidability and complexity analysis by basic paramodulation. *Information and Computation*, 147:1–21, 1998. Extended abstract in IEEE LICS'96.

NR95. Robert Nieuwenhuis and Albert Rubio. Theorem Proving with Ordering and Equality Constrained Clauses. *Journal of Symbolic Computation*, 19(4):321–351, April 1995.

NR97. Robert Nieuwenhuis and Albert Rubio. Paramodulation with Built-in AC-Theories and Symbolic Constraints. *Journal of Symbolic Computation*, 23(1):1–21, May 1997.

NR99a. Robert Nieuwenhuis and José Miguel Rivero. Solved forms for path ordering constraints. In P. Narendran and M. Rusinowitch, editors, *Tenth International Conference on Rewriting Techniques and Applications (RTA)*, LNCS, Trento, Italy, July 2–4, 1999. Springer-Verlag.

NR99b. Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers (to appear), 1999.

PS81. G.E. Peterson and M.E. Stickel. Complete sets of reductions for some equational theories. *Journal Assoc. Comput. Mach.*, 28(2):233–264, 1981.

RV95. Michael Rusinowitch and Laurent Vigneron. Automated deduction with associative commutative operators. *J. of Applicable Algebra in Engineering, Communication and Computation*, 6(1):23–56, 1995.

RW69. G. A. Robinson and L. T. Wos. Paramodulation and theorem-proving in first order theories with equality. *Machine Intelligence*, 4:135–150, 1969.

Vig94. Laurent Vigneron. Associative Commutative Deduction with constraints. In Alan Bundy, editor, *12th International Conference on Automated Deduction*, LNAI 814, pages 530–544, Nancy, France, June 1994. Springer-Verlag.

Wei97. Christoph Weidenbach. SPASS—version 0.49. *Journal of Automated Reasoning*, 18(2):247–252, April 1997.

Towards an Automatic Analysis of Security Protocols in First-Order Logic

Christoph Weidenbach

Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany
weidenb@mpi-sb.mpg.de

Abstract. The Neuman-Stubblebine key exchange protocol is formalized in first-order logic and analyzed by the automated theorem prover SPASS. In addition to the analysis, we develop the necessary theoretical background providing new (un)decidability results for monadic first-order fragments involved in the analysis. The approach is applicable to a variety of security protocols and we identify possible extensions leading to future directions of research.

1 Introduction

The growing importance of the internet causes a growing need for security protocols that protect transactions and communication. It turns out that the design of such protocols is highly error-prone. Therefore, a variety of different methods have been described that analyze security protocols to discover flaws. The topic of this paper is to add a further, new method that is based on automated theorem proving in first-order logic.

In the context of first-order automated theorem proving, Schumann (1997) implemented the well-known BAN logic (Burrows, Abadi & Needham 1990) in first-order logic and then used the automated theorem prover SETHEO to search for proofs in the BAN logic. The BAN logic is a modal belief logic, suitable to express the beliefs of parties in the course of a protocol execution. The logic has been successfully used to analyze authenticity properties of several classical protocols. The BAN logic is not very-well suited for reasoning about secrecy, e.g., possible actions of an intruder. For this purpose finite state (model checking) methods, see, e.g., the overview article by Mitchell (1998), turned out to be successful. Independently from the specific formalization used in such an approach, the protocol is eventually described by a finite model, usually guaranteeing decidability of the investigated properties. The inductive method due to Paulson (1997) uses inductive definitions for actions of the various parties, message sequences etc. as the basis for an analysis. The analysis is supported by the generic, interactive higher-order logic theorem prover Isabelle. Due to the expressiveness of the logic, a detailed modeling of protocols is possible, at the price that explicit induction proofs are usually not automatic.

Our approach tries to combine the benefits of the finite state analysis and the inductive method. The idea is to use fragments of first-order logic that are expressive enough to have infinite, inductive models, but that are still subject to automated theorem proving. For example, our theory is expressive enough to model an intruder that can send all – in general infinitely many (see page 318) – syntactically composable messages. In Section 2 we demonstrate our approach by an analysis of the Neuman & Stubblebine (1993) key exchange protocol. The protocol is translated into first-order monadic Horn fragments. We show that the automated theorem prover Spass (Weidenbach, Afshordel, Brahm, Cohrs, Engel, Keen, Theobalt & Topic 1999) can be successfully used to automatically prove security properties of the protocol and to detect potential errors of an implementation. The Spass input files of the analysis are available at

`http://spass.mpi-sb.mpg.de/`

For space limitations, the analysis of the Neuman-Stubblebine protocol presented here leaves out the second part of the protocol that serves subsequent authentication. This part of the protocol can also be successfully analyzed with our techniques, though. In Section 3 we investigate the Horn fragments involved in the analysis and prove that several of these are decidable in general. The paper ends with a discussion of the achieved results and pointers to future research, Section 4.

2 The Neuman–Stubblebine Protocol

The example protocol we want to study is the key-exchange protocol due to Neuman & Stubblebine (1993). The goal of this protocol is to establish a secure key K_{ab} between two principals A and B that already share secure keys K_{at} and K_{bt} with a trusted server T, respectively.

$$\begin{aligned}
&\text{(i) } A \to B : A, N_a\\
&\text{(ii) } B \to T : B, E_{K_{bt}}(A, N_a, T_b), N_b\\
&\text{(iii) } T \to A : E_{K_{at}}(B, N_a, K_{ab}, T_b), E_{K_{bt}}(A, K_{ab}, T_b), N_b\\
&\text{(iv) } A \to B : E_{K_{bt}}(A, K_{ab}, T_b), E_{K_{ab}}(N_b)
\end{aligned}$$

The protocol starts with A sending the clear-text message (i) to B consisting of two components: A's name and a nonce N_a created by A. So "," in the protocol description means message composition. Nonces are fresh numbers (e.g., random numbers) that are used to prevent replay attacks. Having received this message, B sends the three part message (ii) to the server T. The message starts with B's name, an encrypted middle part and ends with a nonce N_b generated by B. Encryption is denoted by an expression $E_{key}(message)$. The message A, N_a, T_b is encrypted by the secure key K_{bt} that B and T share and contains A's name, the nonce N_a and a time span T_b. The span T_b suggests the expiration time for the eventually generated session key between A and B, an aspect of the protocol that we will not study. The server T decrypts the instructions from B using the key K_{bt} and generates a session key K_{ab} for A and B. Then he sends the three part message (iii) to A. Using the secure key K_{at} the server T encrypts B's name, A's initial nonce N_a, the generated session key K_{ab} and the expiration time span T_b. The second part contains A's name, the session key K_{ab} and T_b encrypted with key K_{bt} and the third part is the nonce N_b. The principal A receives the message, she uses her key K_{at} to decrypt the first part, verifies her nonce N_a, stores the session key K_{ab} and then forwards the second part of T's message to B and adds B's nonce N_b encrypted with the session key K_{ab}, message (iv). Now B decrypts the first part of message (iv) with his key K_{bt}, extracts the session key K_{ab} and by decrypting the second part of the message using the session key he ensures the correct identity of A.

In the sequel, we develop a formalization of this protocol in monadic first-order Horn logic. We are particularly interested in a monadic Horn formalization, because Horn clauses provide a nice minimal model semantics and we are able to provide decidability results for a variety of monadic Horn theories. This will be explained in more detail in the next section, Section 3. We introduce the necessary symbols (predicates, functions, constants) in the course of subsequent message formalization. We adhere to the usual first-order notation where the used operators are $\neg$ (negation), $\wedge$ (conjunction), $\supset$ (implication), $\forall$ (universal quantification), $\exists$ (existential quantification). The key idea of the approach is to formalize the set M of messages that are sent during the execution of the protocol. This model is realistic, because typically such a protocol takes place in an asynchronous framework without any globally available clock. The initial setup for principal A together with message (i) is represented by the formulae

(1) $Ak(key(at, t))$
(2) $P(a)$
(3) $M(sent(a, b, pair(a, na))) \wedge Sa(pair(b, na))$

where *key*, *sent* and *pair* are function symbols, a, b, t, na, at are constant symbols and Ak, P and M are predicate symbols. We use the convention to name function symbols, constant symbols and variables by lowercase letters and predicate symbols start with an uppercase letter. In particular, variables always start with one of the letters *u*–*z*. Formula (1) expresses that A holds the key at for the server T. Formula (2) defines a to be one of the parties of the protocol. Formula (3) states that A sends message (i) (see page 315) and stores that she sent the message. A term $sent(x, y, z)$ means that message z is sent by x to y. The predicate M holds for all sent messages, P holds for all principals and predicates named $\langle Principal\rangle k$ hold all keys for *Principal*, cf. formula (1). Finally, Sa is A's local store that will eventually be used to verify her nonce in message (iii).

Principal B is only interested in fresh nonces. Hence, the formalization of his initial setup and his action seeing A's message (i), formula (3), is

(4) $Bk(key(bt, t))$
(5) $P(b)$
(6) $Bf(na)$
(7) $\forall xa, xna\, [(M(sent(xa, b, pair(xa, xna))) \wedge Bf(xna)) \supset$
$(Sb(pair(xa, xna)) \wedge$
$M(sent(b, t, triple(b, nb(xna), encr(triple(xa, xna, tb(xna)), bt)))))]$

where we formalized B to react properly on any message having the structure of message (i), without knowing the principal in advance, formula (7). So our formalization of the protocol is not a priori restricted to three parties or exactly one execution. The premise $Bf(xna)$ declares that xa's nonce is fresh to B, for otherwise B stops the protocol, since the premise of the above implication becomes false. The initial nonce of A to be fresh to B, formula (6). Furthermore, B is a principal, formula (5), and holds the secure key bt with t, formula (4). The functions nb and tb in formula (7) compute B's nonce and expiration time, respectively. Assuming freshness, these functions only depend on xa's fresh nonce xna. Therefore, for every fresh nonce, B will generate different random numbers and expiration spans. So far, we only defined A's nonce na to be fresh. At the end of this section we will model an intruder that has infinitely many numbers available that are fresh to B. Finally, B stores that he got the key request from xa in his local store Sb to verify his time stamp in the final message (iv).

On seeing message (ii), generated by the succedent of formula (7), the server T sends message (iii), formula (10):

(8) $Tk(key(at, a)) \wedge Tk(key(bt, b))$
(9) $P(t)$
(10) $\forall xb, xnb, xa, xna, xbet, xbt, xat, xk$

$$\begin{array}{l}[(M(sent(xb, t, triple(xb, xnb, encr(triple(xa, xna, xbet), xbt)))) \wedge \\ \quad Tk(key(xbt, xb)) \wedge Tk(key(xat, xa))) \\ \qquad\qquad\qquad\qquad \supset \\ M(sent(t, xa, triple(encr(quadr(xb, xna, kt(xna), xbet), xat), \\ \qquad\qquad encr(triple(xa, kt(xna), xbet), xbt), xnb)))]\end{array}$$

Formula (8) declares the respective keys the server holds for A and B. Formula (9) makes the server a principal. First, in formula (10), the server checks whether he owns the secure key xbt for principal xb. Having the key xbt the server can decrypt the third part of message (ii) and checks whether he also has a secure key xat for communication with principal xa. If all this is satisfied, the server generates a session key $kt(xna)$ by applying the key generation function kt to the nonce xna and sends message (iii) to xa. If xna is a fresh nonce, $kt(xna)$ is a fresh key.

Principal A sees the server message, and tries to decrypt the first part of the message using the secure key at she shares with the server, the antecedent of formula (11). If this succeeds, she checks from her store Sa that this part of the message starts with xb and the nonce she initially sent to xb. Then A forwards the second part xm to xb and encrypts, using the new session key xk contained in her encrypted part, xb's nonce xnb and sends it to xb. In addition, A now owns the session key xk for communication with xb.

(11) $\forall xnb, xbet, xk, xm, xb, xna$

$$\begin{array}{l}[(M(sent(t, a, triple(encr(quadr(xb, xna, xk, xbet), at), xm, xnb))) \wedge \\ \quad Sa(pair(xb, xna)))) \\ \qquad\qquad\qquad\qquad \supset \\ (M(sent(a, xb, pair(xm, encr(xnb, xk)))) \ \wedge \ Ak(key(xk, xb)))]\end{array}$$

Finally, formula (12), B decrypts the first part of the message he received from xa, checks whether it contains his expiration time $xbet$ and uses the session key xk to check whether the second part of the message contains his nonce in the context of xa and xna which he stored in Sb. If all this succeeds, B accepts xk as a secure session key for xa.

(12) $\forall xbet, xk, xnb, xa, xna$
$$[(M(sent(xa, b, pair(encr(triple(xa, xk, tb(xna)), bt), encr(nb(xna), xk)))) \wedge Sb(pair(xa, xna))) \supset Bk(key(xk, xa))]$$

This finishes the formalization of the Neuman-Stubblebine protocol and we now start to analyze it using Spass. First, we want to verify that the protocol yields the desired result, a key between A and B. To this end we saturate the formulae (1)–(12). Recall that the complete below analysis is available via ftp, see Section 1.

Fact 1 Spass *(finitely) saturates the formulae (1)–(12) in less than one second.*

All computations with Spass were performed on a Sun Sparc Ultra 10 with a 300MHz processor running Solaris. Spass computes a finite minimal model that consists in addition to the ground atoms already contained in the formalization of the ground atoms

1. $Bk(key(kt(na), a))$
2. $M(sent(a, b, pair(encr(triple(a, kt(na), tb(na)), bt), encr(nb(na), kt(na)))))$
3. $Ak(key(kt(na), b))$
4. $M(sent(t, a, triple(encr(quadr(b, na, kt(na), tb(na)), at), encr(triple(a, kt(na), tb(na)), bt), nb(na))))$
5. $M(sent(b, t, triple(b, nb(na), encr(triple(a, na, tb(na)), bt))))$
6. $M(sent(a, b, pair(a, na)))$
7. $Sb(pair(a, na))$
8. $Sa(pair(b, na))$

In the next section, Section 3, we will explain what saturation means and how this model is in fact automatically computed by Spass. The model contains exactly the messages that A, B and T communicate according to the protocol. Atoms 1 and 3 indicate the established key $kt(na)$ between A and B. This can be automatically extracted.

Fact 2 Spass *proves the conjecture* $\exists x\, [Ak(key(x, b)) \wedge Bk(key(x, a))]$ *with respect to the saturated theory (Fact 1) in less than one second.*

So, we automatically proved a first important property of the protocol: The protocol terminates and it establishes a key between A and B.

The rest of this section is devoted to an analysis to what extent an intruder can disturb the protocol and how such attacks can be prevented using appropriate implementations for A, B and T. We use the following assumptions for the intruder: First, the intruder can record all sent messages. Second, the intruder cannot break any secure key. In particular, he cannot break the initial keys at and bt. Third, the intruder can send messages and can forge the sender of a message. Fourth, the intruder has no access to the local stores Sa and Sb. The first assumption is formalized by the formula

(13) $\forall xa, xb, xm\, [M(sent(xa, xb, xm)) \supset Im(xm)]$

where $Im(xm)$ means that xm is a message recorded or composed by the intruder. We allow the intruder to decompose messages that are not encrypted

(14) $\forall u, v\, [Im(pair(u, v)) \supset (Im(u) \wedge Im(v))]$

(15) $\forall u, v, w\, [Im(triple(u, v, w)) \supset (Im(u) \wedge Im(v) \wedge Im(w))]$

(16) $\forall u, v, w, z\, [Im(quadr(u, v, w, z)) \supset (Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z))]$

and to newly compose and send these messages in an arbitrary way.

(17) $\forall u, v\, [(Im(u) \wedge Im(v)) \supset Im(pair(u, v))]$

(18) $\forall u, v, w\, [(Im(u) \wedge Im(v) \wedge Im(w)) \supset Im(triple(u, v, w))]$

(19) $\forall u, v, w, x\, [(Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(x)) \supset Im(quadr(u, v, w, x))]$

(20) $\forall x, y, u\, [(P(x) \wedge P(y) \wedge Im(u)) \supset M(sent(x, y, u))]$

So far, the intruder can only decompose messages, rearrange and send them in an arbitrary way. The next two formulae allow the intruder to consider anything he records/composes as a key for anybody and to encrypt messages that way.

(21) $\forall v, w\, [(Im(v) \wedge P(w)) \supset Ik(key(v, w))]$

(22) $\forall u, v, w\, [(Im(u) \wedge Ik(key(v, w)) \wedge P(w)) \supset Im(encr(u, v))]$

So $Ik(key(v, w))$ holds if the intruder considers v to be a key for principal w. Next we tried SPASS to saturate the formulae (1)–(22).

Fact 3 SPASS *does not terminate on saturating the formulae (1)–(22).*

An analysis (by hand) of the produced clauses shows that the formulae (1)–(22) generate infinitely many clauses of the form

$$\forall u\, [Im(u) \supset M(sent(t, a, triple(encr(quadr(b, kt^{i}(na), kt^{i+1}(na), tb(na)), at), \\ encr(triple(a, kt^{i+1}(na), tb(na)), bt), u)))]$$

where $kt^i(na)$ abbreviates the i-fold application of kt to na. In terms of the protocol this corresponds to the following potentially infinite sequence of messages:

$$\begin{array}{l}
I : B \rightarrow T : : B, E_{K_{bt}}(A, K^0_{ab}, T_b), N_b \\
T : T \rightarrow A : : E_{K_{at}}(B, N_a, K^1_{ab}, T_b), E_{K_{bt}}(A, K^1_{ab}, T_b), N_b \\
I : B \rightarrow T : : B, E_{K_{bt}}(A, K^1_{ab}, T_b), N_b \\
T : T \rightarrow A : : E_{K_{at}}(B, N_a, K^2_{ab}, T_b), E_{K_{bt}}(A, K^2_{ab}, T_b), N_b \\
\vdots
\end{array}$$

where the first column shows the real sender of the message (the intruder I fakes the sender to be B) and K^i_{ab} is the i^{th} key generated by T with $K^0_{ab} = K_{ab}$. Thus, the server can be used by the intruder to generate arbitrarily many messages of the form $E_{K_{bt}}(A, K^i_{ab}, T_b)$. Since the intruder knows part of the clear-text message (the principal A) and he knows that only the key K^i_{ab} differs in all these messages and he can get as many messages of this form as he needs, he may be able to start a known-plaintext attack to the protocol. What may be really crucial here is that the intruder can get as many messages of the above format as he may need to break the key K_{bt} (accordingly for K_{at}).

Fact 4 *The non-termination of* SPASS *on formulae (1)–(22) indicates a potential attack to the protocol.*

We can get rid of this attack by a modification to the server T that causes him to reject his own keys as nonces. The (modified) formulae are

$$\begin{array}{l}(10)'\ \forall xb, xnb, xa, xna, xbet, xbt, xat, xk\\ \quad [(M(sent(xb, t, triple(xb, xnb, encr(triple(xa, xna, xbet), xbt)))) \wedge\\ \quad\ Tk(key(xbt, xb)) \wedge Tk(key(xat, xa)) \wedge Nonce(xna))\\ \qquad\qquad \supset\\ \quad M(sent(t, xa, triple(encr(quadr(xb, xna, kt(xna), xbet), xat),\\ \qquad\qquad encr(triple(xa, kt(xna), xbet), xbt), xnb)))]\end{array}$$

(23) $Nonce(na)$

(24) $\forall x\, \neg Nonce(kt(x))$

(25) $\forall x\, [Nonce(tb(x)) \wedge Nonce(nb(x))]$

We give the modified set of formulae to SPASS.

Fact 5 SPASS *terminates on saturating the formulae (1)–(9),(10)′,(11)–(25) in less than one second.*

The minimal model generated by SPASS is infinite, contains the finite minimal model computed out of the formulae (1)-(12) and is described by 64 clauses. For example, it still contains clauses like $Im(x), Im(y) \rightarrow Im(pair(x, y))$, formula (17), that cause the set of potential intruder messages in the minimal model to be infinite. This clause together with formula (20) and one formula out of (9), (2), (5) implies that the set of sent messages is infinite in the minimal model, too

Now we want to check whether the intruder is able to break the protocol. To this end we give SPASS the conjecture $\exists x\, [Ik(key(x, b)) \wedge Bk(key(x, a))]$ expressing that the intruder owns a key for B which B assumes to be a secure key for A.

Fact 6 SPASS *proves the conjecture* $\exists x\, [Ik(key(x, b)) \wedge Bk(key(x, a))]$ *in less than one second.*

The proof indicates a potential attack to the protocol and it was based on the minimal model generated before (Fact 5). By an inspection of the proof it can be seen that the nonce na is the key shared by the intruder and B to communicate with A. With respect to the original protocol and following the proof found by SPASS, the intruder sends instead of message (iv) the message $E_{K_{bt}}(A, N_a, T_b), E_{K_{N_a}}(N_b)$ to B, where he knows N_a from message (i) and N_b, $E_{K_{bt}}(A, N_a, T_b)$ from message (ii). So, without breaking the keys at or bt, the intruder can break the protocol, if nonces can be confused with keys. We can also get rid of this attack by a refinement to B's behavior on A's message (iv), where B does not accept nonces as keys.

$$\begin{array}{l}(7)'\ \forall xbet, xk, xnb, xa, xna\\ \quad [(M(sent(xa, b, pair(encr(triple(xa, xk, tb(xna), bt),\\ \qquad\qquad encr(nb(xna), xk)))) \wedge\\ \quad Sb(pair(xa, xna) \wedge Key(xk)) \quad \supset \quad Bk(key(xk, xa))]\end{array}$$

(26) $\forall x\, \neg[Key(x) \wedge Nonce(x)]$

(27) $\forall x\, Key(kt(x))$

Fact 7 SPASS *terminates on saturating the formulae (1)–(6), (7)′, (8), (9), (10)′, (11)–(27) in less than one second. With respect to the saturation,* SPASS *disproves the conjecture* $\exists x, y, z\, [Ik(key(x, y)) \wedge Bk(key(x, z))]$ *in less than one second.*

Note that we have generalized the conjecture of Fact 6 as we now proved in Fact 7 that the intruder I and B do not share any key at all. The same can be proved by SPASS for possible keys between A and the intruder or the server T and the intruder.

Finally, we also allow the intruder to generate infinitely many nonces that are fresh to B. This enables him to enter the protocol from the very beginning.
(28) $\mathit{If}(\mathit{ni})$
(29) $\forall x\, [\mathit{If}(x) \supset \mathit{If}(\mathit{nif}(x))]$
(30) $\forall x\, [\mathit{If}(x) \supset (\mathit{Bf}(x) \wedge \mathit{Im}(x))]$
The predicate *If* holds for all fresh intruder nonces that are generated by application of *nif* to the initial fresh nonce *ni*, formulae (28), (29). All fresh intruder nonces are also fresh to B and can be used by the intruder to compose messages, formula (30).

Fact 8 SPASS *terminates on saturating the formulae (1)–(6), (7)′, (8), (9), (10)′, (11)–(30) in less than one second. With respect to the saturation,* SPASS *disproves the conjecture* $\exists x, y, z\, [\mathit{Ik}(\mathit{key}(x, y)) \wedge \mathit{Bk}(\mathit{key}(x, z))]$ *in less than one second.*

So this extension to the possibilities of the intruder does not cause an additional attack. In summary, our analysis showed that the protocol terminates, establishes a secure key between A and B and that an intruder cannot break the protocol as long as nonces are not confused with keys.

3 Monadic Horn Theories

In this section we study the monadic Horn theories involved in the previous section. We adhere to the usual definitions for variables, terms, substitutions, atoms, (positive and negative) literals, multisets, and clauses. We give just the most important definitions for our purpose.

The function *vars* maps terms, atoms, literals, clauses and sets of such objects to the set of variables occurring in these objects. A term t is called *shallow* if t is a variable or is of the form $f(x_1, \ldots, x_n)$ where the x_i are not necessarily different. A term t is called *linear* if every variable occurs at most once in t. It is called *semi-linear* if it is a variable or of the form $f(t_1, \ldots, t_n)$ such that every t_i is semi-linear and whenever $\mathit{vars}(t_i) \cap \mathit{vars}(t_j) \neq \emptyset$ we have $t_i = t_j$ for all i, j.

A *clause* is a multiset of literals. We denote clauses by implications of the form $\Gamma \rightarrow \Delta$ where the multiset Γ contains all atoms occurring negatively in the clause and Δ contains all atoms occurring positively in the clause. We abbreviate $\{A\} \cup \Gamma$ by A, Γ for some atom A.

An (Herbrand) *interpretation* I is a set of ground atoms. For any predicate symbol P, we define $I(P) = \{(t_1, \ldots, t_n) \mid P(t_1, \ldots, t_n) \in I\}$. A ground clause $\Gamma \rightarrow \Delta$ is satisfied by I if $\Gamma \not\subseteq I$ or $\Delta \cap I \neq \emptyset$. A non-ground clause is satisfied by I if all its ground instances are satisfied by I. If a clause C is satisfied by I we also say that I is a *model* for C and write $I \models C$. An interpretation is a model for a set of clauses N ($I \models N$), if it is a model for all $C \in N$. A model

I is *minimal* for some set of clauses N, if there is no model J with $J \subset I$ and $J \models N$. The model relation $\models$ can be extended to first-order formulae in the usual way.

A *Horn clause* is a clause with at most one positive literal. A *monadic Horn theory* is a set of Horn clauses where all occurring predicates are monadic. A *declaration* is a clause $S_1(x_1), \ldots, S_n(x_n) \rightarrow S(t)$ with $\{x_1, \ldots, x_n\} \subseteq \mathit{vars}(t)$. It is called a *term declaration* if t is not a variable and a *subsort declaration* otherwise. A subsort declaration is called *trivial* if $n = 0$. A term declaration is called *shallow* (*linear*, *semi-linear*) if t is shallow (linear, semi-linear). Note that shallow term declarations do not include arbitrary ground terms. However, any ground term declaration can be equivalently represented, with respect to the minimal model semantics defined below, by finitely many shallow term declarations. A *sort theory* is a finite set of declarations. It is called *shallow* (*linear*, *semi-linear*) if all term declarations are shallow (linear, semi-linear).

Let N be a sort theory. Then we define the interpretation T^N recursively as follows: (i) for every declaration $\rightarrow S(t) \in N$, substitution σ such that $S(t)\sigma$ is ground, we define $S(t)\sigma \in T^N$ (ii) for every ground substitution σ, declaration $S_1(x_1), \ldots, S_n(x_n) \rightarrow S(t) \in N$, if $S_i(x_i)\sigma \in T^N$, for all $1 \leq i \leq n$, $t\sigma$ ground, then $S(t)\sigma \in T^N$. It is well-known that T^N is the minimal Herbrand model for a sort theory N.

If N is a sort theory then the *first-order theory over* N is the set of all first-order formulae using only predicate and function symbols occurring in N. The first-order theory over N is *decidable*, if we can decide $T^N \models \phi$ for any formula ϕ in the first-order theory over N.

An *atom ordering* is a well-founded, total ordering on ground atoms. Given an atom ordering $\succ$, we will call an atom A maximal with respect to a multiset of atoms Γ, if for any B in Γ we have $B \not\succ A$. Any atom ordering $\succ$ is extended to an ordering on literals by taking the multiset extension of $\succ$ and by identifying any positive literal A with the singleton $\{A\}$ and any negative literal $\neg A$ with the multiset $\{A, A\}$. With this definition, $\neg A$ is greater than A, but is smaller than any literal B or $\neg B$ with $B \succ A$. The multiset extension of the literal ordering induces an ordering on ground clauses. Let us also use $\succ$ to denote both the extension to literals and clauses of any given atom ordering $\succ$. The clause ordering is compatible with the atom ordering; if the maximal literal in C is greater than the maximal literal in D then $C \succ D$. These notions are lifted to the non-ground level as usual: For two non-ground atoms A, B we define $A \succ B$ if $A\sigma \succ B\sigma$ for all ground instances $A\sigma$, $B\sigma$. We say that a Horn clause $\Gamma \rightarrow A$ is *reductive for* the positive literal A, if A is the maximal literal with respect to Γ.

A *selection function* assigns to each (ground) clause a possibly empty set of occurrences of negative literals. If C is a clause and *sel* a selection function then the literal occurrences in $\mathit{sel}(C)$ are called *selected*. In particular, $\mathit{sel}(C) = \emptyset$ indicates that no literal is selected.

Definition 1. *An inference by* ordered resolution (with selection) *between two Horn clauses takes the form*

$$\frac{\mathit{\Gamma} \rightarrow A \qquad B, \mathit{\Lambda} \rightarrow \mathit{\Delta}}{\mathit{\Gamma}\sigma, \mathit{\Lambda}\sigma \rightarrow \mathit{\Delta}\sigma}$$

such that (i) σ is the most general unifier between A and B (ii) $\mathit{\Gamma}\sigma \rightarrow A\sigma$ is reductive for $A\sigma$, (iii) no literal is selected in $\mathit{\Gamma}$, and (iv) B is selected, or else no literal is selected in $B, \mathit{\Lambda} \rightarrow \mathit{\Delta}$ and $B\sigma$ is maximal in $B\sigma, \mathit{\Lambda}\sigma \rightarrow \mathit{\Delta}\sigma$. The multiset $\mathit{\Delta}$ is either empty or contains exactly one atom.

The inference rule *sort resolution* we employ in this paper is an instance of ordered resolution. Using an appropriate option setting SPASS implements sort resolution with respect to the theories considered in the previous section. The used ordering is any atom ordering satisfying $T(s) \succ S(t)$ if s contains t as a proper subterm. For example, a Knuth-Bendix ordering where all function and predicate symbols have weight one has this property. Given a clause $C = S_1(t_1), \ldots, S_i(t_n) \rightarrow S(t)$ the selection function *sor* for sort resolution is defined by $S_i(t_i) \in \mathit{sor}(C)$ if (i) t_i is a non-variable term or (ii) all t_j are variables and t_i is a variable that does not occur in t or (iii) all t_j are variables occurring in t and t is a variable.

Let $\succ$ be a total atom ordering and *sel* a selection function. Given a set of ground clauses N, we use induction with respect to $\succ$ to define a Herbrand interpretation I_C and a set E_C, for each clause C in N, as follows.

Definition 2. *Let I_C be the set $\bigcup_{C \succ D} E_D$. Furthermore, $E_C = \{A\}$ if (i) $C = \mathit{\Gamma} \rightarrow A$ is reductive for A, (ii) $\mathit{\Gamma} \subseteq I_C$ and (iii) $A \notin I_C$. Otherwise, E_C is the empty set.*

If $E_C = \{A\}$, we also say that C *produces* A and call C a *productive clause.* Finally, by I, we denote the Herbrand interpretation $\bigcup_{C \in N} E_C$. Whenever we need to emphasize the dependency of the interpretation I from the particular clause set N, we will use the notation I^N. If N is clause set containing non-ground clauses, then I^N is the interpretation generated by all ground instances of clauses from N. A non-ground clause $C \in N$ is called productive if a ground instance of C is productive for I^N.

A clause C is a *condensation* of a clause D, if C is a proper (unordered) factor of D that subsumes D. A set of clauses N is called *saturated* if it is closed under condensation, the deletion of subsumed clauses and any clause generated by an ordered resolution inference from clauses from N is subsumed by some clause in N. Ordered resolution in general allows more powerful notions of simplification/redundancy (Bachmair & Ganzinger 1994), but for the purpose of this paper subsumption and condensation suffices.

Corollary 1 (Bachmair & Ganzinger (1994)). *Let N be a set of Horn clauses saturated by ordered resolution. Then either N contains the empty clause or I^N is a model for N.*

Lemma 1. *Let N be a monadic Horn theory saturated by sort resolution that does not contain the empty clause. Then $I^N = T^{N'}$ where N' is the set of all term declarations and trivial subsort declarations occurring in N.*

Proof. First, we show by contradiction that all productive clauses in N are either term declarations or trivial subsort declarations. This implies $I^N = I^{N'}$. So assume $C\sigma = S_1(t_1)\sigma, \ldots, S_n(t_n)\sigma \to S(t)\sigma$ is the minimal (with respect to $\succ$) ground instance of a clause $C \in N$ that produces $S(t)\sigma$ but some $S_i(t_i)$ is selected in C. Since $C\sigma$ produces $S(t)\sigma$, we know $\{S_1(t_1)\sigma, \ldots, S_n(t_n)\sigma\} \subseteq I^N_{C\sigma}$ and $S(t)\sigma \notin I^N_{C\sigma}$. So there is a ground clause $D\tau = T_1(s_1)\tau, \ldots, T_m(s_m)\tau \to S_i(s)\tau$, $D \in N$, $S_i(s)\tau = S_i(t_i)\sigma$ where $D\tau \prec C\sigma$. No literal in D is selected (we chose $C\sigma$ to be minimal) and therefore we can generate a sort resolution resolvent R from C and D. This resolvent has a ground instance $R\lambda = (S_1(t_1), \ldots, S_{i-1}(t_{i-1}), T_1(s_1)\tau, \ldots, T_m(s_m)\tau, S_{i+1}(t_{i+1}), \ldots, S_n(t_n) \to S(t))\sigma$, $R\lambda \prec C\sigma$, is productive and produces $S(t)\sigma$ contradicting that $C\sigma$ produces $S(t)\sigma$. Therefore, every productive clause in N has no selected literal and is hence a term declaration or a trivial subsort declaration.

Second, we show $I^{N'} \subseteq T^{N'}$ by induction on the clause ordering. If $S(t)\sigma \in I^{N'}$ then there is a productive clause $S_1(x_1), \ldots, S_n(x_n) \to S(t) \in N'$. If $n = 0$ then by definition of $T^{N'}$, case (i), we have $S(t)\sigma \in T^{N'}$. If $n \neq 0$ then by definition of $I^{N'}$ we know $\{S_1(x_1), \ldots, S_n(x_n)\}\sigma \subseteq I^{N'}$ and therefore by induction hypothesis $\{S_1(x_1), \ldots, S_n(x_n)\}\sigma \subseteq T^{N'}$. Now, by definition of $T^{N'}$, case (ii), we have $S(t)\sigma \in T^{N'}$.

Third, we show $T^{N'} \subseteq I^{N'}$ by structural induction on the definition of $T^{N'}$. If $S(t)\sigma \in T^{N'}$ is generated by some clause $\to S(t) \in N'$ then $\to S(t)\sigma$ is productive and hence $S(t)\sigma \in I^{N'}$. If $S(t)\sigma \in T^{N'}$ is generated by some clause $C = S_1(x_1), \ldots, S_n(x_n) \to S(t) \in N'$ we know $\{S_1(x_1), \ldots, S_n(x_n)\}\sigma \subseteq T^{N'}$ and hence, by induction hypothesis, $\{S_1(x_1), \ldots, S_n(x_n)\}\sigma \subseteq I^{N'}$. Furthermore, C is a term declaration and by definition of our ordering C is reductive for $S(t)$. Hence, $S_1(x_1)\sigma, \ldots, S_n(x_n)\sigma \to S(t)\sigma$ is productive and generates $S(t)\sigma \in I^{N'}$.

Lemma 2. *Let N be a semi-linear sort theory. Then the first-order theory over N is decidable.*

Proof. First, we transform N into a shallow sort-theory N'. We recursively replace every declaration $S_1(x_1), \ldots, S_n(x_n) \to S(f(t_1, \ldots, t_n))$ where t_i is not a variable by two new declarations

$$S_{m_1}(x_{m_1}), \ldots, S_{m_l}(x_{m_l}), R(y) \to S(f(s_1, \ldots, s_n))$$
$$S_{j_1}(x_{j_1}), \ldots, S_{j_k}(x_{j_k}) \to R(t_i)$$

where y and R are new, $s_j = t_j$ if $t_j \neq t_i$, $s_j = y$ if $t_j = t_i$ for all $1 \leq j \leq n$, $vars(t_i) = \{x_{j_1}, \ldots, x_{j_k}\}$ and $vars(f(t_1, \ldots, t_n)) \setminus vars(t_i) = \{x_{m_1}, \ldots, x_{m_l}\}$. Since $f(t_1, \ldots, t_n)$ is semi-linear, t_i and $f(s_1, \ldots, s_n)$ are semi-linear as well. The transformation terminates generating a shallow sort theory N' and by an induction argument it can be proved that $T^N(P) = T^{N'}(P)$ for any (monadic) predicate P occurring in N. Second, Weidenbach (1998) showed that any shallow sort theory N' can be transformed into a $Rec_=$ tree automaton (Bogaert & Tison 1992) such that for any monadic predicate P the language accepted by the *tree automaton in the state corresponding to P is exactly $T^{N'}(P)$. Third, Comon & Delor (1994) showed that the first-order theory over $Rec_{=}$ automata can be decided.*

Theorem 9. *Let N be a monadic Horn theory finitely saturated by sort resolution. If all term declarations in N are semi-linear, then the first-order theory over the productive clauses in N is decidable.*

Proof. By Lemma 1 and Lemma 2.

The technique used to prove Theorem 9 is a combination of results obtained in the context of saturation based theorem proving with results developed in the area of finite tree automata. In particular, saturation based theorem proving cannot be a priori used to decide the first-order theory for some finitely saturated semi-linear sort theory. One problem is that Skolemization cannot be carried out in the usual way, since existential quantifiers must not introduce new symbols but have to be interpreted over the minimal model of the sort theory. The fragment for conjectures we used in the previous section is indeed decidable by saturation based methods in general.

Lemma 3. *Let N be a semi-linear sort theory. Then the first-order fragment $\exists x_1, \ldots, x_n\, [A_1 \wedge \ldots \wedge A_k]$ from the first-order theory over N where all A_i are atoms can be decided by sort resolution.*

Proof. The formula $\exists x_1, \ldots, x_n\, [A_1 \wedge \ldots \wedge A_k]$ holds in T^N iff $\forall x_1, \ldots, x_n\, [\neg A_1 \vee \ldots \vee \neg A_k]$ does not hold in T^N. This can be checked by sort resolution saturating the set $N \cup \{A_1, \ldots, A_k \rightarrow\}$. This was shown to be decidable by Jacquemard, Meyer & Weidenbach (1998).

The above Lemma 3 explains the success of Spass on the queries tested for Fact 2, Fact 6 and Fact 7 in Section 2. The saturated theory, Fact 1, is a ground theory and therefore semi-linear. This ground theory is the result of saturating the clauses resulting from the formulae (1)–(12), see Section 2, by sort resolution. By deleting all non-productive clauses, the remaining clauses are exactly the atoms shown on page 318 plus the ground atoms that are already contained in the formulae (1)–(12). Hence, Spass can in fact decide the query of Fact 2. The saturated theories that are the basis for the conjectures investigated in Fact 6 and Fact 7 are syntactically not semi-linear but contain non-linear variable occurrences at different depth. However, all these occurrences are restricted by monadic predicates having a finite extension in the minimal model. Lemma 3 also holds for this extension.

Saturating a sort theory extended by a clause $A_1, \ldots, A_k \rightarrow$ is a process closely related to sorted unification (Weidenbach 1998). This is even decidable for sort theories extended by certain, restricted forms of equations as shown by Jacquemard et al. (1998). What remains to be shown is in which cases saturation of a monadic Horn theory terminates.

Lemma 4. *Let N be a monadic Horn theory where all positive literals are linear and shallow. Then N can be finitely saturated by sort resolution and the productive clauses of the saturated theory form a linear shallow sort theory.*

Proof. Since any clause where no negative literal is selected is either a trivial subsort declaration or a linear shallow term declaration, any application of sort resolution takes one of the following three forms:

$$\frac{S_1(x_{i_1}), \dots, S_k(x_{i_k}) \rightarrow S(f(x_1, \dots, x_n)) \qquad S(f(t_1, \dots, t_n)), \Lambda \rightarrow \Delta}{S_1(x_{i_1})\sigma, \dots, S_k(x_{i_k})\sigma, \Lambda \rightarrow \Delta}$$

where $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, $\{x_{i_1}, \dots, x_{i_k}\} \subseteq \{x_1, \dots, x_n\}$ *or*

$$\frac{S_1(x_{i_1}), \dots, S_k(x_{i_k}) \rightarrow S(f(x_1, \dots, x_n)) \qquad S(y), \Lambda \rightarrow \Delta}{S_1(x_{i_1}), \dots, S_k(x_{i_k}), \Lambda\tau \rightarrow \Delta\tau}$$

where $\tau = \{y \mapsto f(x_1, \dots, x_n)\}$, *again* $\{x_{i_1}, \dots, x_{i_k}\} \subseteq \{x_1, \dots, x_n\}$ *or*

$$\frac{\rightarrow S(x) \qquad S(t), \Lambda \rightarrow \Delta}{\Lambda \rightarrow \Delta}$$

For all three types of inferences the following invariant holds: All non-variable terms in the resolvent are proper subterms of the right parent clause or the resolvent contains at most one linear shallow non-variable term. Since N contains only finitely many different terms and only monadic predicates, there are only finitely many resolvents that can be generated with respect to subsumption and condensation. Hence, the saturation terminates. Furthermore, for every type of a resolution inference, if the resolvent contains a positive atom $S(t)$ then t is a linear shallow term. Hence, all productive clauses in the saturated theory are either trivial subsort declarations or linear shallow term declarations and therefore form a linear shallow sort theory.

The restrictions on the terms required for Lemma 4 are close to the border of non-termination (undecidability). If we only require declarations to be linear, then even if we additionally require that every literal is linear, the saturation process does not terminate, in general. This can be shown by a reduction of the ground word problem for word equations to this problem. We encode equality by a two place function symbol e and add one monadic predicate T for "truth". Then all axioms of equality, except reflexivity, result monadic Horn clauses with linear literals. The congruence axioms result in linear clauses because all function symbols (except e itself) are monadic. Reflexivity would result in a non-linear clause. However, it is sufficient for the reduction to code reflexivity for ground terms, i.e, the occurring constants, resulting in linear clauses. Finally, any word equation can be transformed into a linear Horn clause by expressing non linearities through linear disequations. For example, the word equation $ab \approx a$ that corresponds to the term equation $a(b(x)) \approx b(x)$ is expressed by the linear monadic Horn clause $T(e(b(x), y)) \rightarrow T(e(a(b(x)), y))$.

If we only require the positive literals in the Horn theories of Lemma 4 to be shallow, the resulting theories can also not be finitely saturated, in general. We can simply reduce the undecidable unifiability problem for arbitrary sort theories to this problem. We code an arbitrary term declaration $S_1(x_1), \dots, S_n(x_n) \rightarrow$

$S(t)$ by $S_1(x_1), \ldots, S_n(x_n), T(f(y,t)) \rightarrow S(y)$ where y, f and T are new and add the shallow declaration $\rightarrow T(f(x,x))$.

With respect to the saturation results in Section 2, Lemma 4 partly explains the behavior of SPASS. All considered sets of formulae in Section 2, have positive occurrences of non-linear atoms and Fact 3 shows that the suggested techniques are in general not strong enough to guarantee termination. However, the theory we used in the previous section for the intruder, is a theory where all positive atoms are linear and shallow. So, by Lemma 4, this theory can always be finitely saturated. Note that this theory is independent of the investigated protocol as long as it can be represented by a set of messages. Furthermore, this theory can still be finitely saturated if we add finitely many ground term declarations. The productive clauses of the saturation of the initial protocol, Fact 1, solely contains ground declarations. Hence, the non-termination of the saturation of the protocol theory extended by the intruder theory indicates an infinite number of messages "exchanged" by the protocol and the intruder, the messages we isolated below Fact 3. Note that none of the above termination results for sort resolution holds for standard resolution or ordered resolution without our particular selection strategy.

4 Conclusion

This paper consists of two major parts: First, it shows that using automated theorem proving techniques security protocols like the Neumann-Stubblebine protocol can be successfully analyzed. The suggested techniques apply to a variety of protocols, e.g, further key exchange protocols (see Schneier (1996)). In particular, we can finitely model an "infinite" intruder. Second, we have been able to prove that parts of the used first-order fragments can be decided in general. This includes more sophisticated properties than the properties we tested for the Neumann-Stubblebine protocol. The decidability results are also useful in a more general context. For example, Lemma 4 offers a new fine grained approximation for Horn programs (Charatonik, McAllester, Niwinski, Podelski & Walukiewicz 1998).

So far our formalization mainly models "reachability", i.e., that certain situations can(not) occur. For many protocols "liveness" properties, e.g., that a certain situation will definitely occur, require a more sophisticated formalization. We already successfully analyzed signature exchange protocols with respect to such properties using SPASS. In order to explain the termination of SPASS on these experiments, the results of Section 3 have to be extended to the non-Horn equality case. So an important direction of future research is to extend the decidability results with respect to the needed fragments. The results of the experiments with SPASS we did so far support that this should be possible. Furthermore, our results indicate that it should be possible to design an automatic tool for the analysis of security protocols that is not restricted to a finite state model.

Acknowledgments: I'm indebted to Harald Ganzinger, Andreas Nonnengart, Birgit Pfitzmann and Matthias Schunter for numerous useful comments and discussions. The comments of the reviewers helped a lot in improving the quality of the paper.

References

Bachmair, L. & Ganzinger, H. (1994), 'Rewrite-based equational theorem proving with selection and simplification', *Journal of Logic and Computation* **4**(3), 217–247.

Bogaert, B. & Tison, S. (1992), Equality and disequality constraints on direct subterms in tree automata, *in* A. Finkel & M. Jantzen, eds, 'Proceedings of 9th Annual Symposium on Theoretical Aspects of Computer Science, STACS92', Vol. 577 of *LNCS*, Springer, pp. 161–171.

Burrows, M., Abadi, M. & Needham, R. (1990), 'A logic of authentication', *ACM Transactions on Computer Systems* **8**(1), 18–36.

Charatonik, W., McAllester, D., Niwinski, D., Podelski, A. & Walukiewicz, I. (1998), The horn mu-calculus, *in* 'Proceedings 13th IEEE Symposium on Logic in Computer Science, LICS'98', IEEE Computer Society Press, pp. 58–69.

Comon, H. & Delor, C. (1994), 'Equational formulae with membership constraints', *Information and Computation* **112**, 167–216.

Jacquemard, F., Meyer, C. & Weidenbach, C. (1998), Unification in extensions of shallow equational theories, *in* T. Nipkow, ed., 'Rewriting Techniques and Applications, 9th International Conference, RTA-98', Vol. 1379 of *LNCS*, Springer, pp. 76–90.

Mitchell, J. C. (1998), Finite-state analysis of security protocols, *in* A. J. Hu & M. Y. Vardi, eds, 'Computer Aided Verification (CAV-98) : 10th International Conference', Vol. 1427 of *LNCS*, Springer, pp. 71–76.

Neuman, B. C. & Stubblebine, S. G. (1993), 'A note on the use of timestamps as nonces', *ACM SIGOPS, Operating Systems Review* **27**(2), 10–14.

Paulson, L. C. (1997), Proving properties of security protocols by induction, *in* J. Millen, ed., 'Proceedings of the 10th IEEE Computer Security Foundations Workshop', IEEE Computer Society, pp. 70–83.

Schneier, B. (1996), *Applied Cryptography*, 2 edn, Wiley.

Schumann, J. (1997), Automatic verification of cryptographic protocols with setheo, *in* 'Proceedings of the 14th International Conference on Automated Deduction, CADE-14', Vol. 1249 of *LNAI*, Springer, Townsville, Australia, pp. 87–100.

Weidenbach, C. (1998), Sorted unification and tree automata, *in* W. Bibel & P. H. Schmitt, eds, 'Automated Deduction - A Basis for Applications', Vol. 1 of *Applied Logic*, Kluwer, chapter 9, pp. 291–320.

Weidenbach, C., Afshordel, B., Brahm, U., Cohrs, C., Engel, T., Keen, E., Theobalt, C. & Topic, D. (1999), System description: Spass version 1.0.0, *in* H. Ganzinger, ed., '16th International Conference on Automated Deduction, CADE-16', LNAI, Springer. This volume.

A Confluent Connection Calculus

Peter Baumgartner[1], Norbert Eisinger[2], and Ulrich Furbach[1]

[1] Universität Koblenz-Landau
Institut für Informatik
{peter,uli}@uni-koblenz.de
[2] Universität München
Institut für Informatik
eisinger@informatik.uni-muenchen.de

Abstract. This work[1] is concerned with basic issues of the design of calculi and proof procedures for first-order connection methods and tableaux calculi. Proof procedures for these type of calculi developed so far suffer from not exploiting proof confluence, and very often unnecessarily rely on a heavily backtrack oriented control regime.
As a new result, we present a variant of a connection calculus and prove its *strong* completeness. This enables the design of backtrack-free control regimes. To demonstrate that the underlying fairness condition is reasonably implementable we define an effective search strategy.

1 Introduction

This work is concerned with basic issues of the design of calculi and proof procedures for first-order connection methods and tableaux calculi. Calculi we have in mind include connection calculi [6], first-order clausal tableaux with rigid variables [9], more recent developments like A-ordered tableaux [14,12], and tableaux with selection function [13]. Let us refer to all these calculi by the term “rigid variable methods”. Recently complexity issues for those kinds of calculi have been considered in [19].

We propose a new technique for the design of proof procedures for rigid variable methods. The proposed technique should also be applicable to calculi which *avoid* rigid variables in the first place, like SATCHMO [16], MGTP [10], hyper tableaux [4] and ordered semantic hyper linking [17]. Usually the price for getting around rigid variables in these approaches is that they involve some uninformed ground instantiation in special cases. These calculi are likely to profit from techniques enabling them to handle rigid variables as well.

A more recent development is the disconnection method [7]. Unlike the calculi mentioned above, and like our calculus below, it avoids blind instantiation by the use of unification. Unlike our method and unlike the free variable methods mentioned above, the disconnection method does not deal with “free variables”[2]. In free variable methods, a branch or a path is considered as “solved”

[1] A full version of this article, which includes all proofs and other details omitted here, is available as [3]

[2] This is not meant to disqualify the disconnection method; merely, we want to point out differences.

if it contains a pair (K, L) of literals that are complementary; in the disconnection method, the respective "\$-closed" condition on (K, L) is that K and L are equal after instantiating all variables with the same (new) constant \$. For instance, $(P(X, Y), \neg P(Z, Z))$ is considered as complementary. The disconnection method has inference rules to further instantiate clauses derived so far, but the parent clauses of an inference step have to be kept. This instantiation is driven by connections that are not \$-closed. Since substitutions are applied locally to clauses, a previously \$-closed connection may become open again. If the clause containing $\neg P(Z, Z)$ is further instantiated to, say, $\neg P(a, a)$ this implies that the connection $(P(X, Y), \neg P(a, a))$ is no longer \$-closed. In free variable methods this effect is avoided because, clearly, complementary literals *remain* complementary after instantiation; there is no need to check previously closed paths after instantiation whether they are still closed. On the other hand, for the disconnection method only one variant per clause needs to be kept; this is not possible in free variable methods.

In sum, we feel that theses approaches are rather different. A more rigid and systematic comparison would be interesting future work.

Our approach is based on the observation that current proof procedures for rigid variable methods rely on the following weak completeness theorem:

> *Weak completeness:* a clause set S is unsatisfiable if and only if *there is* a derivation from S which is also a refutation.

The search space thus is the space of *derivations*; it requires a tentative control regime such as backtracking, which explores *all* possible derivations.

Proof confluent calculi like the ones mentioned at the beginning of the introduction, however, should admit a *strong* completeness theorem of the form:

> *Strong completeness:* a clause set S is unsatisfiable if and only if *every* (fair) derivation from S is a refutation.

Consequently, proof procedures following this theorem can do with an irrevocable control regime that needs to develop only *one single* derivation and may safely ignore alternatives as it proceeds. They can thus *reuse* information which would be lost in a backtracking intensive approach. Typically they enumerate *models* but not *derivations* (the hyper tableaux calculus [4] is an example that enumerates models, model elimination [15] is an example for the enumeration of derivations).

Put abstractly, the source to gain efficiency is that there are usually many derivations for the *same* model, and all but one derivation can be avoided. In this paper, we will develop a strong completeness result for a modified connection calculus (the CCC calculus, Section 2) together with first steps towards a respective proof procedure.

This result closes a strange gap: the connection calculus in [6] is proof confluent on the propositional level, but its first-order version is not. This is the only calculus we are aware of having this property. Although other free-variable methods, such as the first-order tableaux calculus in Fitting's book [9] *are* proof-confluent, they are *implemented* as if they were non-confluent. This yields unnecessary inefficiencies, and the main motivation for the work presented in this paper is to find a cure for them.

Prawitz Procedures. Let us first briefly recall the basic idea of current proof procedures for rigid variable methods and identify the tackled source of inefficiency.

"Usual" proof procedures like the one used in the 3TAP prover [11], the LeanTAP prover [5]), the connection procedure proposed in [6], and the one in Fitting's Book [9] follow an idea suggested by Prawitz [18] and can be seen as more or less direct translation of the following formulation of the Herbrand-Skolem-Gödel theorem (we can restrict our attention to clause logic here):

> A clause set S is unsatisfiable if and only if there is a finite set S' of variants of clauses of S and there is a substitution δ such that $S'\delta$ is unsatisfiable, where $S'\delta$ is viewed as a propositional clause set.

Now, in accordance with the theorem, proof procedures for rigid variable methods typically realise the following scheme to prove that given clause set $S = \{C_1, \ldots, C_n\}$ is unsatisfiable. Following Voronkov [19], we call it *The Procedure*:

Procedure 1 (The Procedure).

(i) Let $\mu = 1$ called *multiplicity*.
(ii) Let $S^\mu = \{C_1^1, \ldots, C_1^\mu, \ldots, C_n^1, \ldots, C_n^\mu\}$ be a set of pairwise variable-disjoint clauses, such that C_i^j is a variant of C_i for $1 \le i \le n$, $1 \le j \le \mu$. It is usual to call S^μ an *amplification* of S.
(iii) Check if there is a substitution δ such that $S^\mu\delta$ is propositionally unsatisfiable.
(iv) If such a δ exists, then stop and return "unsatisfiable"; otherwise let $\mu = \mu+1$ and go to step (ii).

Completeness of *The Procedure* is achieved by, first, a fairness condition in the generation of the amplifications S^μ in step (ii), namely by uniformly taking $1, 2, 3, \ldots$ variants of every clause in S in round $1, 2, 3, \ldots$, and, second, by exhaustively searching for substitutions in step (iii).

Connection Methods. How is step (iii) in *The Procedure* realised? Our primary interest is in connection methods (also called matrix methods), hence we will briefly recall the idea: define a matrix to be any set of quantifier free formulae. For our purposes it suffices to consider clause sets only. Notice that S^μ is a matrix. A *path* through a matrix is obtained by taking exactly one literal from every clause. A connection is a pair of literals, which can be made complementary by application of a substitution; with each connection we associate a most general unifier σ achieving this. In order to realise step (iii), proof procedures for connection calculi search for a substitution δ such that every path through $S^\mu\delta$ contains a pair of complementary literals (we say that δ *closes* S^μ). If such a δ

exists S must be unsatisfiable: take some arbitrary ground instance of $S^\mu\delta$ and observe that this set is propositionally unsatisfiable, because any possible way to satisfy a conjunction of disjunctions is excluded by virtue of the complementary literals.

For δ, it is sufficient to search through the *finite* space of most general unifiers making literals along paths (or branches) complementary (this guarantees the termination of step (iii). See e.g. [6] for a concrete procedure to decide if a δ exists which renders all paths complementary. For our purposes it suffices to rephrase the underlying idea: let $p_1, p_2, \ldots, p_n$ be any enumeration of all paths through the current amplification. It can be shown by usual lifting techniques that there is δ which simultaneously renders all paths as complementary if and only if there is a sequence $p_1\delta_1,\ p_2\delta_1\delta_2, \ldots, p_n\delta_1\cdots\delta_n$, where δ_i is a most general unifier associated with a connection in $p_i\delta_1\cdots\delta_{i-1}$. In other words, by defining $\delta := \delta_1\cdots\delta_n$ one recognises that δ can be computed incrementally. Notice, however, that the "there is" quantification is to be translated in a backtracking-oriented procedure.

Example 1 (Connection Method). Consider the following unsatisfiable clause set:

$$S = \{P(X) \vee Q(X),\ \ \neg P(c),\ \ \neg P(a) \vee \neg P(b),\ \ \neg Q(a),\ \ \neg Q(b)\}$$

We write a matrix as a vertical sequence of clauses. Hence S^1 and S^2 look like this[3]:

S^1:
$$\begin{array}{c} P(X^1) \vee Q(X^1) \\ \neg P(c) \\ \neg P(a) \vee \neg P(b) \\ \neg Q(a) \\ \neg Q(b) \end{array}$$

S^2:
$$\begin{array}{c} P(X^1) \vee Q(X^1) \\ P(X^2) \vee Q(X^2) \\ \neg P(c) \\ \neg P(a) \vee \neg P(b) \\ \neg Q(a) \\ \neg Q(b) \end{array}$$

A path is obtained by traversing the matrix from top to bottom, picking up one literal from every clause.

The Procedure starts with $\mu = 1$, i.e. with S^1. By looking at the path through S^1 which passes through $P(X^1)$ one recognises that there are three candidate MGUs $\delta_1^1 = \{X^1/a\}$, $\delta_2^1 = \{X^1/b\}$ and $\delta_3^1 = \{X^1/c\}$. Since none of $S^1\delta_1^1$, $S^1\delta_2^1$ and $S^1\delta_3^1$ is propositionally unsatisfiable, *The Procedure* has to consider S^2. An incremental computation of a substitution δ which closes S^2 might proceed as follows: it starts by considering the connection $(P(X^1), \neg P(c))$ which results in $\delta_1^2 = \{X^1/c\}$. The next connection would be $(Q(X^2), \neg Q(a))$ with MGU $\delta_2^2 = \{X^2/a\}$. The combined substitution $\delta_1^2\delta_2^2 = \{X^1/c, X^2/a\}$ does not close the matrix, neither will $\delta_1^2\delta_3^2$, where $\delta_3^2 = \{X^2/b\}$. Hence, backtracking occurs until eventually the "right" substitution $\delta = \{X^1/a,\ X^2/b\}$ is computed, which closes S^2.

[3] In the literature, matrices are also written horizontally.

Proof Search with Tableaux. Typically, the search for δ in step (iii) is organised as the construction of a free-variable tableau where a branch in a tableau stands for a path through the current amplification.

For space efficiency reasons it is prohibitive to explicitly represent in step (iii) all paths through S^{μ} in memory. For example, only 15 clauses consisting of 3 literals result in more than 14 million paths. A respective tableau thus has in the worst case the same number of branches. Hence, one possibility for step (iii) is to only keep in memory one path p (or branch in a tableau) at a time. A closing substitution δ_p is guessed, and if δ_p cannot be extended to a substitution δ which simultaneously closes all paths, then backtracking occurs and a different candidate δ_p is guessed. If all that fails, then a completely new tableau construction is started when entering step (iii) in the next round.

We emphasise that this is a common idea in all proof procedures for free-variable tableaux and connection calculi we are aware of. It is an intrinsic weakness and it seems that it is not solved by the many improvements that are around for tableau and connection calculi now.

2 A Confluent Connection Calculus

In this section we indroduce the confluent version of a connection calculus. For this we briefly have to set up the usual prerequisites. After indroducing the calculus together with an abstract notion of fairness, we show how this fairness can be realised and finally we prove strong completeness.

2.1 Preliminaries

For a literal L we denote by $|L|$ the atom of L, i.e. $|A| = A$ and $|\neg A| = A$ for any atom A. Literals K and L are *complementary* iff K and L have different sign and $|L| = |K|$.

By var(*object*) we denote the set of variables occurring in *object*, where *object* is a term, a literal, a clause or a set of one of these. Our notion of a *substitution* is the usual one [1], and, as usual, we confuse a substitution with its homomorphic extension to terms, literals, clauses and clause sets. For a substitution σ, we denote by $\mathrm{dom}(\sigma)$ the (finite) set $\{X \mid X\sigma \neq X\}$, by $\mathrm{cod}(\sigma)$ the set $\{X\sigma \mid X \in \mathrm{dom}(\sigma)\}$ and by $\mathrm{vcod}(\sigma)$ the set $\mathrm{var}(\mathrm{cod}(\sigma))$. Substitution γ is a *ground* substitution iff $\mathrm{vcod}(\gamma) = \emptyset$; it is a *ground substitution for object* iff additionally $\mathrm{var}(object) \subseteq \mathrm{dom}(\gamma)$. A substitution σ is idempotent, if $X\sigma\sigma = X\sigma$ for each variable X. This is the case iff $\mathrm{dom}(\sigma) \cap \mathrm{vcod}(\sigma) = \emptyset$.

A clause is a finite disjunction of literals. In the following, S is the given finite input clause set, and M is a matrix for S, i.e. a set of clauses, each of which is an instance of a clause from S. It is worth emphasizing that in a matrix we consider clauses *not* to be individually universally quantified. That is, all variables are free, and thus applying a substitution, say, $\{Y/a\}$ to the matrix $\{\neg P(Y), P(Y) \vee Q(Y)\}$ would affect all occurrences of Y.

A *connection with substitution* σ is a pair of literals (L, K) such that $L\sigma$ and $K\sigma$ are complementary. A *connection* is a pair of literals that is a connection with some substitution. In these definitions, we replace "substitution σ" by "MGU σ" only if additionally $\sigma = \text{unify}(\{|K|, |L|\})$, where unify returns any most general unifier of its argument. Below we make use of the following assumption:

Assumption 1. If a set Q of atoms is unifiable, then unify(Q) returns an idempotent most general unifier σ with (i) $\text{dom}(\sigma) \subseteq \text{var}(Q)$ and (ii) $\text{vcod}(\sigma) \subseteq \text{var}(Q)$.

Notice that this is a very mild assumption: (i) says that σ operates on the variables of Q only, and (ii) says that σ must not introduce new variables. Clearly, this is satisfied by any "standard" unification procedure.

A *path* through a matrix M is a set of literals, obtained by taking exactly one literal from each clause of M. A path is *closed* iff it contains a pair of complementary literals, otherwise it is *open*. A matrix is *open* iff there is a (at least one) path through it that is open, otherwise it is *closed*.

Notice that there is exactly one path through the "empty" matrix $\{\}$, which is the empty set $\{\}$ of literals; notice further that this path is open. On the other hand, if a matrix M contains the empty clause, then there is no path through M, in particular no open path, and the matrix is closed.

2.2 The Calculus

We are going to define the calculus *CCC* (Confluent Connection Calculus). The constituents are the inference rules, the notion of a derivation, and a fairness condition for derivations.

Definition 1 (CCC Inference Rules and Derivation). The inference rule *variant step on M* is defined as follows:

$$\frac{M}{M \cup \mathsf{new}(S)} \quad \text{if} \begin{cases} \text{1. } \mathsf{new}(S) \text{ contains exactly one variant of each clause of } S\text{, and} \\ \text{2. the members of } \mathsf{new}(S) \text{ are pairwise variable disjoint and} \\ \quad \text{each of them is variable disjoint with each clause in } M. \end{cases}$$

The inference rule *connection step on (K, L) in M* is defined as follows:

$$\frac{M}{M \cup M\sigma} \quad \text{if} \begin{cases} \text{1. there are clauses } C \in M \text{ and } D \in M \text{ such that } C = K \vee R_K, \\ \quad D = L \vee R_L, \text{ for some clauses } R_K \text{ and } R_L, \text{ and} \\ \text{2. } \{K, L\} \subseteq p \text{ for some open path } p \text{ through } M\text{, and} \\ \text{3. } (K, L) \text{ is a connection with MGU } \sigma. \end{cases}$$

The set $(M \cup M\sigma) \setminus M$ is called the set of *new clauses* (of this inference step). If conditions 1 to 3 above hold for M and (K, L), we say that a connection step is *applicable* to (K, L) in M or that (K, L) is a *candidate* for a connection step in M.

We say that a connection step on (K, L) in M is *progressive* if $M \cup M\sigma \neq M$ (i.e. at least one clause in the conclusion is new). If $M \cup M\sigma = M$ we say that it is *non-progressive.*

Note that a connection step on a connection (K, L) with $|K| = |L|$ is impossible and therefore neither progressive nor non-progressive – any path containing $\{K, L\}$ would be closed, contradicting condition 2. above.

Any sequence $D = (\{\} = M_0), M_1, \ldots, M_n, \ldots$ is called a *derivation from* S, provided that M_{i+1} is obtained from M_i, which is open, by a single application of one of the inference rules (for $i \geq 0$). A *refutation* is a derivation that contains (i.e. ends in) a closed matrix.

Notice that we do not have inference rules that delete clauses. Thus, every derivation has the following *chain property*: $(\{\} = M_0) \subseteq M_1 \subseteq \cdots \subseteq M_n \subseteq \cdots$. The inclusions are not strict, because non-progressive steps are allowed as well (though they are not needed at all).

2.3 Example

Suppose the given input clause set is $S = \{\neg P(X),\ \ P(Y) \vee Q(Y),\ \ \neg Q(Z)\}$. Clearly, S is unsatisfiable. We develop a refutation.

We have to start with the empty matrix M_0. Only a variant step can be applied to M_0, hence M_1 is just a copy of S (the left matrix in the figure below). For each matrix M_i we discuss the possibilities for one open path through M_i, which is indicated by underlining.

$$\begin{array}{llr|llr}
M_1: & \underline{\neg P(X)} & (C_1) & M_2: & \underline{\neg P(X)} & (C_1) \\
 & \underline{P(Y)} \vee Q(Y) & (C_2) & & P(X) \vee \underline{Q(X)} & (C_2\sigma_1) \\
 & \underline{\neg Q(Z)} & (C_3) & & \underline{\neg Q(Z)} & (C_3) \\
 & & & & P(Y) \vee \underline{Q(Y)} & (C_2)
\end{array}$$

The underlined path in M_1 is open, and we carry out a connection step on $(\neg P(X), P(Y))$. Suppose that unify returns $\sigma_1 = \{Y/X\}$. This step is progressive and results in the matrix M_2 (right side in the figure above).

Now there are two connections in the underlined open path in M_2 to which a connection step is applicable: $(Q(X), \neg Q(Z))$ and $(\neg Q(Z), Q(Y))$. By applying a connection step to the former we would obtain a closed matrix, thus ending the refutation. In order to make the example more illustrative, let us instead apply a connection step to $(\neg Q(Z), Q(Y))$ with MGU $\sigma_2 = \{Y/Z\}$. This gives us matrix M_3, which is depicted below.

The underlined path in M_3 offers three possibilities. First, the connection $(Q(X), \neg Q(Z))$ is still a candidate for a connection step, but we disregard it for the same reason as in the previous matrix. Second, the connection step on $(\neg Q(Z), Q(Y))$ with MGU $\{Y/Z\}$ would not be progressive and therefore not interesting (if unify returned $\{Z/Y\}$ instead, the step would be progressive, though). Third, a connection step on $(\neg P(X), P(Z))$ is progressive no matter which of the two MGUs unify returns. Suppose that this step is applied with $\sigma_3 = \{Z/X\}$. The resulting matrix M_4 is depicted in the figure below as well. Note that $P(X) \vee Q(X)$ occurs twice in M_4. Since matrices are sets, deleting one of these occurrences would represent the same set.

$$\begin{array}{lcr|lcr}
M_3: & \underline{\neg P(X)} & (C_1) & M_4: & \neg P(X) & (C_1) \\
 & P(X) \vee \underline{Q(X)} & (C_2\sigma_1) & & P(X) \vee Q(X) & (C_2\sigma_1) \\
 & \underline{\neg Q(Z)} & (C_3) & & \neg Q(X) & (C_3\sigma_3) \\
 & \underline{P(Z)} \vee Q(Z) & (C_2\sigma_2) & & P(X) \vee Q(X) & (C_2\sigma_2\sigma_3) \\
 & P(Y) \vee \underline{Q(Y)} & (C_2) & & P(Y) \vee Q(Y) & (C_2) \\
 & & & & \neg Q(Z) & (C_3) \\
 & & & & P(Z) \vee Q(Z) & (C_2\sigma_2)
\end{array}$$

M_4 is closed because either path through the first three clauses alone is closed. Hence we have found a refutation.

2.4 Fairness

Control strategies for any kind of rule-based system usually depend on some notion of fairness, when several rules or rule instances are applicable and one of them has to be selected for the next step. In matrix M_2 above, two connections were candidates for progressive connection steps, and one of them was selected to derive M_3, while the other was disregarded. In M_3 the disregarded connection was again a candidate for a progressive connection step and was again disregarded. Fairness is a condition on the choices made by the strategy to prevent that an applicable step is disregarded forever.

In many cases, especially with control strategies for logical calculi, fairness can be defined very simply as *exhaustiveness*: every step that is applicable to any state will ultimately be performed. There are standard text book procedures to implement exhaustive strategies effectively; in particular, resolution calculi can be treated this way.

This simple approach is sufficient if the underlying rules are commutative in the following sense: whenever two steps are applicable in some state, each of them remains applicable in the successor state produced by the other. If, on the other hand, the rules are not commutative, then the application of one step might destroy the applicability of the other. Such a phenomenon makes exhaustiveness an inherent impossibility and turns fairness into a more difficult question.
Unfortunately, our system is of the non-commutative variety. As an example let

$$M_i = \{\underline{\neg P(a)} \vee R, \quad \underline{P(X)}, \quad \underline{\neg Q(a)}, \quad \underline{Q(Y)} \vee P(Y), \quad \ldots\}$$

where the underlined path through M_i is open. Both connections $(\neg P(a), P(X))$ and $(\neg Q(a), Q(Y))$ are candidates for progressive connection steps in M_i. Disregarding the former and applying the latter yields

$$M_{i+1} = \{\neg P(a) \vee R, \quad P(X), \quad \neg Q(a), \quad Q(a) \vee P(a), \quad Q(Y) \vee P(Y), \quad \ldots\} \ .$$

Now any path through M_{i+1} containing the disregarded connection $(\neg P(a), P(X))$ is closed. This is a consequence of the presence of the clauses

$\neg Q(a)$ and $Q(a) \vee P(a)$ in M_{i+1}. Therefore a connection step on the first connection, which would introduce the new clause $P(a)$, is no longer applicable – the second condition in the definition of a connection step can no longer be satisfied for this connection. Note that other paths through M_{i+1} are still open, though.

By the way, had we selected the other applicable step in M_i, the next matrix would have been

$$M'_{i+1} = \{\neg P(a) \vee \underline{R}, \quad \underline{P(a)}, \quad \underline{P(X)}, \quad \underline{\neg Q(a)}, \quad \underline{Q(Y)} \vee P(Y), \quad \ldots\}$$

in which the disregarded connection $(\neg Q(a), Q(Y))$ is still a candidate for a progressive connection step as shown by the underlined open path through M'_{i+1}. Selecting this connection next we can achieve that the new clauses of both steps are present.

Should the presence of all of these clauses happen to be indispensable for closing the matrix, it might be that the sequence $M_0, \ldots, M_i, M_{i+1}$ cannot be extended to a refutation whereas the sequence $M_0, \ldots, M_i, M'_{i+1}$ can – this would be a typical case of non-confluence. Fortunately, it turns out that our calculus *is* confluent: in all such situations, either none or both sequences can be continued to a refutation. The example illustrates why this is not a trivial property.

Coming back to fairness, the simple exhaustiveness definition is not possible, and we need a more complex definition. With this definition we will prove below our main result, that any fair derivation from an unsatisfiable clause set is a refutation (strong completeness).

Definition 2 (Fairness). A derivation $D = M_0, M_1, \ldots, M_i, \ldots$ is *fair* iff it is a refutation or else the following two conditions are satisfied:

1. For every $i \geq 0$ there is a $j \geq i$ such that M_{j+1} results from M_j by a variant step.
2. For every $i \geq 0$ and every connection (K, L) with MGU σ that is a candidate for a progressive connection step in M_i there is a $j \geq i$ such that one of the following is true:
   (a) $M_i\sigma \subseteq M_j$.
   (b) Every path p through M_j with $\{K, L\} \subseteq p$ is closed.

Condition 1 simply requires variant steps to be performed "every now and then".

Condition 2.(a) formalises that any progressive connection step that remains applicable sufficiently long must ultimately be performed. More precisely, the condition does not enforce that this very step be performed, but only that its effect be achieved at some point – regardless whether by this step or by some other.

In a nice case, if Condition 2.(a) holds, the connection (K, L) is irrelevant from M_j onward, because any connection step on (K, L) in some later matrix would result in clauses that have been introduced up to M_j anyway.

However, perhaps contrary to the intuition, this is not always so. The reason is that the MGU σ of the connection, say $\{X/Y\}$, may be applicable to clauses containing the variable X *that are derived only later in the derivation.* Hence, applying σ at a later time may well be progressive. Condition 2.(a) covers this possibility: let the later time be i', then there must again be a time j' at which the clauses that are new for $M_{i'}$ have been introduced.

Condition 2.(b) captures the case that a connection step loses its applicability because of other steps.

2.5 Achieving Fairness

Our notion of fairness – Definition 2 – is defined as an abstract mathematical property derivations may or may not enjoy. In order to implement a proof procedure, however, we need not only the property, but an effective strategy for the construction of a derivation that is guaranteed to have this property.

Fortunately, the existence of such a strategy can be demonstrated with a fairly simple approach: use an iteratively increasing bound T that serves both as a limit for the term nesting depth and as a trigger for the application of variant steps.

More precisely, we call a progressive connection step *T-progressive*, if at least one (but not necessarily all) of its new clauses has a term nesting depth not exceeding T. Building on this, we define an effective strategy as follows (in contrast to *The Procedure*):

Procedure 2 (The CCC Procedure).

(i) Initialise M with the empty matrix and T with the maximum term nesting depth of the clauses in S.
(ii) While M is open repeat:
   (a) Modify M by applying one variant step.
   (b) Increment T.
   (c) Modify M by applying a T-progressive connection step.
   Repeat this until saturation, i.e., until no such step is applicable any more.

The sequence of values of M over time corresponds to the constructed derivation. Let us denote these values by $M_0, M_1, \dots, M_i, \dots$

Note that this strategy is indeed irrevocable in the sense described in the introduction. It never backtracks to previous values of M or makes any other provision for reconsidering alternatives at a later point in time. A subtle difference to *The Procedure* concerns the generated amplifications. One might think that each iteration through (a) creates the next amplification S^μ of S. However, during the application of T-progressive connection steps in phase (c), the current M_i is extended by new clauses yielding M_{i+1}, whereas S^μ in *The Procedure* would only be instantiated, but not extended. In a sense our connection steps combine instantiation with (partial) amplification.

Theorem 1. *Any derivation (from any finite input clause set) constructed by Procedure 2 is fair.*

Of course there are certainly much better strategies. The procedure above is only meant as a proof that fair derivations can be effectively constructed.

What remains, now, is to show that the fairness of derivations, no matter how achieved, ensures confluence. This result is established in the next section.

3 Completeness

Definition 3 (Bounded Downward Closed Clause Set). Let S be a finite clause set, and γ be a ground substitution for S. Define a set of sets $S \downarrow \gamma$ as follows: $S \downarrow \gamma = \{S\delta \mid S\delta\gamma = S\gamma \text{ for some substitution } \delta\}$.

That is, $S \downarrow \gamma$ is the set of instances of S, (including S, variants of S and $S\gamma$ itself) that can further be instantiated to $S\gamma$ by γ itself. Notice that if $S\delta' \in S \downarrow \gamma$ and $S\delta'\delta''\gamma = S\gamma$ for some δ'', then also $S\delta'\delta'' \in S \downarrow \gamma$ by simply taking $\delta = \delta'\delta''$. In this sense $S \downarrow \gamma$ is "downward closed".

Lemma 1. *$S \downarrow \gamma$ is a finite set of finite sets.*

Proposition 1. *Let M be a matrix, p be a path through M, and γ be a ground substitution for M. Suppose that $\{K, L\} \subseteq p$ (for some literals K and L), that (K, L) is a connection with substitution γ, and that (K, L) is a connection with idempotent MGU σ. Then $M\gamma = M\sigma\gamma$.*

Definition 4 (Ordering on Clause Sets). Let M and N be finite clause sets. Define $M \succ N$ iff $\text{var}(M) \supset \text{var}(N)$.

It is easy to see that $\succ$ is an irreflexive and transitive relation, hence a strict (partial) ordering. Obviously, since $\text{var}(M)$ is always finite, it is well-founded as well.

Lemma 2 (Path Extension). *Let $D = M_0, M_1, \ldots, M_n, \ldots$ be an infinite derivation from clause set S. Let M be a finite clause set, and suppose that $M \subseteq M_k$ for some k. Then there is a path p through M such that for every $i \geq k$ there is an open path p_i through M_i with $p \subseteq p_i$.*

That is, we can find a path p through M such that p will be part of some open path as the derivation proceeds.

The main result of this paper is the following completeness theorem. Note that it assures completeness, whenever fairness is guaranteed; hence we have strong completeness and thus proof confluence.

Theorem 2 (Completeness). *Let S be the given input clause set. If S is unsatisfiable, then every fair derivation from S is a refutation.*

Proof. By Herbrand's theorem there is a finite set S^g of ground instances of clauses from S which is unsatisfiable. It can be presented as a finite set M of pairwise variable disjoint variants of clauses from S and a ground substitution γ such that $M\gamma = S^g$.

Now, let D be a fair derivation from S and assume contrary to the theorem that D is not a refutation.

Since D is fair, the variant rule must be applied infinitely often. Recall that we never delete clauses from matrices. Hence, at some time point, say l, it will thus be that $M \subseteq M_l$ (and hence also $M \subseteq \bigcup_{i \geq 0} M_i$). W.l.o.g. we can assume that each clause in M is syntactically identical to one of the variants introduced up to M_l; otherwise rename M (and γ) appropriately.

Now let $N \in M \downarrow \gamma$ be a minimal set wrt. $\succ$ such that $N \subseteq \bigcup_{i \geq 0} M_i$. We have to check that such a set N exists: since $M \subseteq \bigcup_{i \geq 0} M_i$ and it trivially holds that $M \in M \downarrow \gamma$ it is clear that M itself might be a candidate to be taken as N. Now, since $\succ$ is a well-founded ordering on finite clause sets, and the elements of $M \downarrow \gamma$ are finite (cf. Lemma 1), any chain $(N_0 := M) \succ N_1 \succ \cdots \succ N_{n-1} \succ N_n$ with $N_i \in M \downarrow \gamma$ and $N_i \subseteq \bigcup_{i \geq 0} M_i$ must be finite, and we set $N := N_n$, provided that this chain cannot be extended to the right. Thus, N is minimal wrt. $\succ$ restricted to the elements in $\bigcup_{i \geq 0} M_i$.

The proof technique now is to construct an N' with $N' \in M \downarrow \gamma$ and $N' \subseteq \bigcup_{i \geq 0} M_i$ and $N \succ N'$, contradicting the minimality of N.

Since derivations have the chain property, the property $N \subseteq \bigcup_{i \geq 0} M_i$ implies by the fact that N is a finite set that

$$N \subseteq M_k \text{ , for some } k. \tag{1}$$

Therefore we can apply Lemma 2 and conclude that there is a path p through N such that for every $i \geq k$ there is an open path p_i through M_i with $p \subseteq p_i$. This result will be used further below – the immediate consequence that this p is open will be needed in a moment.

Clearly, $N\gamma$ is an unsatisfiable ground clause set, because $N \in M \downarrow \gamma$ means by definition that $N = M\delta$ for some δ such that $M\delta\gamma = M\gamma$, hence $N\gamma = M\gamma$, and $M\gamma$ was assumed to be ground and unsatisfiable above. Consequently, any path through $N\gamma$ contains a pair of complementary literals. In particular, $p\gamma$ contains a pair of complementary literals, say $K\gamma$ and $L\gamma$, where

$$\{K, L\} \subseteq p \ . \tag{2}$$

In other words, (K, L) is a connection with substitution γ. But then (K, L) is a connection with MGU σ, where σ is computed by unify. Then Proposition 1 is applicable and we conclude that $N\sigma\gamma = N\gamma$. The element $N\sigma$ has the form $N\sigma = M(\delta\sigma)$ because $N = M\delta$. For this element we obtain by the previous equation and by $N\gamma = M\gamma$ the properties $M(\delta\sigma)\gamma = N\gamma = M\gamma$, which is just the definition for

$$N\sigma \in M \downarrow \gamma \ . \tag{3}$$

The next subgoal is to show that $N\sigma$ is strictly smaller than N.

Since p is open and $\{K, L\} \subseteq p$ it trivially holds that $|K| \neq |L|$. On the other hand, σ is a most general unifier of $|K|$ and $|L|$. Hence there is at least one variable, say X, in $\{K, L\} \subseteq p$, for which $X\sigma \neq X$, that is, $X \in \text{dom}(\sigma)$. Now p is a path through N, therefore $X \in \text{var}(N)$.
Obviously, it holds that $\text{var}(N\sigma) \subseteq \text{var}(N) \cup \text{vcod}(\sigma)$. By Assumption 1 $\text{vcod}(\sigma) \subseteq \text{var}(\{K, L\}) \subseteq \text{var}(N)$, hence $\text{var}(N\sigma) \subseteq \text{var}(N)$.
The idempotence of σ implies $\text{dom}(\sigma) \cap \text{vcod}(\sigma) = \emptyset$, thus $X \notin \text{vcod}(\sigma)$ and $X \notin \text{var}(N\sigma)$. So we have $\text{var}(N\sigma) \subset \text{var}(N)$, which means nothing but

$$N \succ N\sigma \ . \tag{4}$$

There remains to be shown that

$$N\sigma \subseteq \bigcup_{i\geq 0} M_i \ . \tag{5}$$

We distinguish two cases, each leading to the conclusion 5.

Case 1: For every $i \geq k$, it is not the case that (K, L) is a candidate for a progressive connection step in M_i.

As concluded above by the application of Lemma 2, for every $i \geq k$ there is an open path p_i through M_i with $p \subseteq p_i$. With the fact that $\{K, L\} \subseteq p$ (obtained in 2 above) it follows that $\{K, L\}$ is a candidate for a connection step in M_i, for every $i \geq k$.

Together with the assumption of this Case 1 we obtain that for every $i \geq k$ a connection step on (K, L) in M_i exists, but this connection step is non-progressive. Hence, for every $i \geq k$, $M_i \cup M_i\sigma = M_i$. But then with $N \subseteq M_k$ as obtained in 1 we get the following chain:

$$N\sigma \ \subseteq \ M_k\sigma \ \subseteq \ \bigcup_{i\geq k}(M_i \cup M_i\sigma) \ = \ \bigcup_{i\geq k} M_i \ \subseteq \ \bigcup_{i\geq 0} M_i \ .$$

Case 2: This is the complement of Case 1. Hence suppose that for some $i \geq k$ the connection (K, L) is a candidate for a progressive connection step in M_i. We are given that the given derivation D is fair (cf. Def. 2). In particular, Condition 2 in the definition of fairness holds wrt. the connection (K, L) and M_i. Let $j \geq i$ be the point in time claimed there.

Assume that Condition 2.(b) is satisfied (this will yield a contradiction). This means that every path p' through M_j with $\{K, L\} \subseteq p'$ is closed. But at the same time, however, observing that $j \geq k$ (since $j \geq i$ and $i \geq k$) we concluded further above by application of Lemma 2 that there is an open path p_j through M_j with $p \subseteq p_j$. Since $\{K, L\} \subseteq p$ (as by 2) and thus $\{K, L\} \subseteq p_j$, we arrive at a contradiction to the just assumed by setting $p' = p_j$.

Hence Condition 2.(b) cannot be satisfied, and consequently Condition 2.(a) must be satisfied. This means that $M_i\sigma \subseteq M_j$. Recall that derivations have the chain property. From $i \geq k$ we thus get $M_k \subseteq M_i$. Together with $N \subseteq M_k$ then $N\sigma \subseteq M_i\sigma$, and so $N\sigma \subseteq M_j$ follows immediately. Clearly, $N\sigma \subseteq \bigcup_{i\geq 0} M_i$ as well. This completes Case 2.

Notice that in both cases we concluded with 5. Altogether, 3, 4, and 5 contradict the minimality of N.

Hence, the assumption that D is not a refutation must be wrong, and the theorem holds. □

4 Conclusions

In this paper we discussed the drawback of naive connection methods, and, as an improvement, defined a confluent connection calculus and *The CCC Procedure* based on this calculus. We gave a strong completeness proof, and hence proved proof confluence as well.

The CCC Procedure treats matrices, which are sets of clauses. Consequently, the order of clauses does not play any role for fairness or completeness. Hence, more refined proof procedures are free to represent sets as (duplicate-free) lists. A good strategy then would be to append new instances of clauses derived during the saturation phase or during variant steps at the end of the list, because then it is cheap to identify and represent those new closed paths that simply extend previously closed paths[4].

Our calculus achieves confluence by taking into account only derivations which obey a fairness condition. This condition is formulated as an abstract formal property of derivations. It allows to formulate a strong completeness theorem, stating that *any fair* derivation leads to a proof, provided that a proof exists at all.

The difficulty was to define the calculus in such a way that an *effective* fairness condition can be stated. Defining an effective fair strategy is much less straightforward than in resolution calculi (CCC is not commutative, unlike resolution).

That it is not trivial was observed already in [8]. There, a rigid-variable calculus is defined and a strong completeness result is proven. However, the question how to define an *effective* fair strategy had to be left open. Thus, our new approach can be seen to address open issues there.

We came up with a strategy, which is based on a term-depth bound and we proved that this strategy indeed results in fair derivations.

We are aware of the fact, that this effective strategy is only a first step towards the design of an efficient proof procedure based on the CCC-calculus. We expect improvements over "usual" tableaux based implementations of connection calculi, which do not exploit confluence.

This article is a first step, a lot of work remains to be done. In particular the saturation step within our strategy for achieving fairness needs to be turned into a more algorithmic version. Another important topic is to avoid the generation of redundant clauses. To this end regularity, as it is implemented in clausal tableaux would be a first attempt. A further point would be to investigate under which conditions the variant inference rule can be dispensed with, or how to modify the calculus so that new variants of input clauses are introduced more sparsely.

Acknowledgments. We are grateful to Donald Loveland for comments on an earlier version.

References

1. F. Baader and K. U. Schulz. Unification Theory. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction. A Basis for Applications.* Kluwer, 1998.
2. P. Baumgartner. Hyper Tableaux — The Next Generation. In H. de Swaart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods.* LNAI 1397, Springer, 1998.

[4] This strategy resembles much a tableau procedure – one that takes advantage of *confluence*, however.

3. P. Baumgartner, N. Eisinger, and U. Furbach. A Confluent Connection Calculus. Fachberichte Informatik 23–98, Universität Koblenz-Landau, Universität Koblenz-Landau, D-56075 Koblenz, 1998.
4. P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, LNAI 1126. Springer, 1996.
5. B. Beckert and J. Posegga. leanT^AP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.
6. W. Bibel. *Automated Theorem Proving.* Vieweg, 2nd edition, 1987.
7. J.-P. Billon. The Disconnection Method. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 1071. Springer, 1996.
8. F. Bry and N. Eisinger. Unit resolution tableaux. Research Report PMS-FB-1996-2, Institut für Informatik, LMU München, 1996.
9. M. Fitting. *First Order Logic and Automated Theorem Proving.* Texts and Monographs in Computer Science. Springer, 1990.
10. H. Fujita and R. Hasegawa. A Model Generation Theorem Prover in KL1 using a Ramified-Stack Algorithm. In *Proc.*8th *of the Eighth International Conference on Logic Programming*, pages 535–548, Paris, France, 1991.
11. R. Hähnle, B. Beckert, and S. Gerberding. The Many-Valued Theorem Prover 3TAP. Interner Bericht 30/94, Universität Karlsruhe, 1994.
12. R. Hähnle and S. Klingenbeck. A-Ordered Tableaux. *Journal of Logic and Computation*, 6(6):819–833, 1996.
13. R. Hähnle and C. Pape. Ordered tableaux: Extensions and applications. In D. Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, LNAI 1227, pages 173–187. Springer, 1997.
14. S. Klingenbeck and R. Hähnle. Semantic tableaux with ordering restrictions. In A. Bundy, editor, *CADE 12*, LNAI 814, pages 708–722. Springer, 1994.
15. D. Loveland. Mechanical Theorem Proving by Model Elimination. *JACM*, 15(2), 1968.
16. R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In E. Lusk and R. Overbeek, editors, *CADE 9*, LNCS 310, pages 415–434. Springer, 1988.
17. D. A. Plaisted and Y. Zhu. Ordered Semantic Hyper Linking. In *Proceedings of AAAI-97*, 1997.
18. D. Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960.
19. A. Voronkov. Strategies in rigid-variable methods. In *IJCAI 97*, Nagoya, 1997.

Abstraction-Based Relevancy Testing for Model Elimination

Marc Fuchs[1] and Dirk Fuchs[2]

[1] Fakultät für Informatik, TU München
80290 München, Germany
fuchsm@informatik.tu-muenchen.de
[2] Fachbereich Informatik, Universität Kaiserslautern
67663 Kaiserslautern, Germany
dfuchs@informatik.uni-kl.de

Abstract. In application areas like formal software verification or in connection with lemma use, the success of automated theorem provers strongly depends on their ability to choose from a huge number of given axioms only some relevant ones. We present a new technique for deleting irrelevant clauses for model elimination proof procedures which is based on the use of abstractions. We analyze general conditions under which abstractions are well-suited for choosing useful clauses and investigate whether certain abstractions are applicable. So-called symbol abstractions are further examined in a case study which, e.g., introduces new techniques for the automatic generation of abstractions. We evaluate our techniques by means of experiments performed with the prover SETHEO.

1 Introduction

Automated theorem proving (ATP) systems have achieved impressive results in many mathematical areas. But normally, these results are obtained in domains where the specifications are nearly 'minimal'. This means that a high degree of the clauses which constitute the proof task are actually 'relevant' for a proof. In many practical application areas like *formal software verification* this is not the case. Normally, hundreds of axioms have to be handled and an ATP system is overwhelmed by the large amount of (mostly irrelevant) data. Similar problems occur when (possibly irrelevant) *lemmas* are used in addition to the given clauses.

Thus, a notion of relevancy and the estimation of whether a clause is relevant for a proof task are essential for the practical success of an automatic prover. The aim is to provide criteria which allow the deletion of certain clauses without affecting the completeness of a prover. An even more refined goal is to delete not only clauses which do not occur in a proof but to delete clauses which do not occur in proofs that can be found quickly by a proof system. Thus, we want to deal with the development of a *search-based* notion of relevancy and with techniques for deleting clauses irrelevant regarding this notion. Since different ATP systems carry out different kinds of searches regarding their calculi (e.g., iterative deepening search or breadth-first search methods) we have chosen one specific calculus and one specific search method. We deal with the *connection tableau calculus* (CTC) and *iterative deepening* search methods.

The first main contribution is a notion of relevancy which considers the iterative deepening method to be applied. Clauses are considered relevant w.r.t. finite parts of the search space defined by a so-called *completeness bound*. Then, we deal with methods for excluding clauses which are not relevant. In contrast to [4,3] where we developed *heuristical* methods for filtering clauses based on so-called *partial evaluation techniques* we now develop *exact* criteria using *abstractions.*

The use of abstractions has a long tradition in the area of automated theorem proving (e.g., [10,11,12,5,1]). Abstractions have been used to guide the search for theorems. Based on proofs of the abstracted proof task (which may be found because of the small size of the abstract search space) the search for a proof is controlled (which was often called *mapping back*). The application of abstractions for relevancy testing for the CTC and iterative deepening search is novel.

We provide an abstraction-based framework for excluding irrelevant clauses from a given theory. Abstractions which satisfy certain conditions can be used for a reduction of the input clause set. We discuss the suitability of prominent abstractions used in the past and newly introduced ones. We identify so-called *symbol abstractions* as well-suited and conduct a detailed case-study regarding the use of symbol abstractions. New principles and methods for an *automatic* generation of abstractions are provided. Furthermore, we introduce *partial mapping-back* techniques as new methods for an improved relevancy test.

The methods are evaluated by means of experiments with the model elimination prover SETHEO [9]. We have experimented with proof tasks which are augmented with lemmas (see [13,4]). We could indeed identify a lot of irrelevant lemmas and thus increased the performance of SETHEO regarding both run time and number of solved problems.

2 Connection Tableau Calculus

A *signature* is a tuple $sig = (\mathcal{P}, \mathcal{F})$ where $\mathcal{P}$ and $\mathcal{F}$ are finite sets of predicate and function symbols with fixed arities. Henceforth, $\mathcal{V}$ is an enumerable set of variables. Terms, atoms, literals, and substitutions over sig and $\mathcal{V}$ are given as usual. The *complement* $\sim l$ of a literal l is $\neg A$ if $l = A$ and A if $l = \neg A$. *Clauses* are disjunctions $l_1 \vee \ldots \vee l_n$ of literals $l_1, \ldots, l_n$. If $C = l_1 \vee \ldots \vee l_n$ the literal l_i is denoted by (C, i). If a clause C' can be obtained from C by renaming the variables of C then C' is called a *variant* of C. If C' can be obtained by permuting the literals of a variant of C, C' is a *permutation variant* of C. C (properly) subsumes D, denoted by $C \unlhd D$ ($C \lhd D$) if there is a substitution σ with $C\sigma \subseteq D$ ($C\sigma \subset D$). We have $l_1 \vee \ldots \vee l_n \subseteq (\subset) s_1 \vee \ldots \vee s_m$ if $\{l_1, \ldots, l_n\} \subseteq (\subset) \{s_1, \ldots, s_m\}$.

2.1 Proof Objects

The CTC works on *connection tableaux* for a set of *input clauses* $\mathcal{C}$. A *tableau* T for $\mathcal{C}$ is a labeled tree (t, λ, ι) where λ maps all non-root nodes of the tree t to literals and ι maps all nodes to sets of literals. λ is called the *literal* and ι the *context lemma* label. Furthermore, if the immediate successors $v_1, \ldots, v_n$ of a node v of T are labeled (by λ) with literals $l_1, \ldots, l_n$, then $l_1 \vee \ldots \vee l_n$ must be an instance of a clause from $\mathcal{C}$. We call $l_1 \vee \ldots \vee l_n$ a *tableau clause*. A tableau is a *connection tableau* if each non-leaf node v has a leaf node v' among its immediate successor nodes such that $\lambda(v)$ is complementary to $\lambda(v')$.

Nodes in a tree t are identified with *positions*. A position p is a finite sequence over $I\!N$ ($p \in I\!N^*$). The node v at the position p in t is denoted by $t|p$. The root node is at the empty position. The n successors of a node $t|p$, $p \in I\!N^*$, are $t|(p.1), \ldots, t|(p.n)$. $Pos(t)$ is the set of positions of t.

A *marked connection tableau* is a pair $T = ((t, \lambda, \iota), \mu)$ where (t, λ, ι) is a connection tableau and μ a partial function from the leaf nodes of t to the set of nodes of t. A branch of T is *open* if the leaf is not marked, otherwise it is *closed*. $s = \lambda(v)$ at the leaf v of an open branch as well as v are called *subgoals*. $T|p$ identifies for $p \neq ()$ the literal $\lambda(t|p)$. $T[p]$ is the tableau clause *below* $t|p$.

Trees t_1 and t_2 are *isomorphic* w.r.t. isomorphism π if π is a bijective mapping from the nodes of t_1 to the nodes of t_2. Moreover, if v is a node in t_1 and $v_1, \ldots, v_n$ are its immediate successors in t_1 then the immediate successors of $\pi(v)$ are $\pi(v_1), \ldots, \pi(v_n)$. $T_1 = ((t_1, \lambda_1, \iota_1), \mu_1)$ and $T_2 = ((t_2, \lambda_2, \iota_2), \mu_2)$ are isomorphic w.r.t. π if t_1 and t_2 are isomorphic w.r.t. π, $\mu(\pi(v)) = \pi(\mu(v))$ for all v in t_1, and there are no additional marks in T_2. $T_1 = ((t_1, \lambda_1, \iota_1), \mu_1)$ is *embedded* in $T_2 = ((t_2, \lambda_2, \iota_2), \mu_2)$ if t_1 is isomorphic to a *contraction* t_2' of t_2 (i.e., t_2 can be obtained from t_2' by attaching subtrees to non-leaf nodes of t_2') w.r.t. isomorphism π. Furthermore, μ_1 is the function with the smallest domain so that $\mu_2(v) = w \wedge \exists v', w'$ with $\pi(v') = v$, $\pi(w') = w$ implies $\mu_1(v') = w'$.

2.2 The Calculus

A connection tableau calculus *calc* is defined by a set of inference rules, a set of search space reduction techniques, and a subgoal selection function.

We introduce the inference rules *start*, *extension* (*ext*), *reduction* (*red*), *factorization* (*fac*), and *folding-up* (*fu*). A *set of inference rules* $\mathcal{I}$ contains at least *start*, *ext*, and *red*. We present the rules in form of transition relations. Let $\mathcal{C}$ be a set of input clauses. Let $T = ((t, \lambda, \iota), \mu)$ be a given marked tableau for $\mathcal{C}$. All rules produce a new marked tableau $T' = ((t', \lambda', \iota'), \mu')$.

We have $T \vdash_{start,C} T'$ if T is the trivial tableau, i.e. one consisting of an unlabeled node v and label (marking) functions with an empty domain. Furthermore, t' is obtained from t by attaching for each of the literals of $C \in \mathcal{C}$ a node below v which is labeled by λ' with the respective literal. μ' has an empty domain. $\iota'(w) = \emptyset$ for each node w of t'. Only *start relevant* clauses are needed (e.g. [9]). Thus, we restrict *start* to some *start clauses* $\mathcal{S} \subseteq \mathcal{C}$ where $\mathcal{S}$ contains at least one start relevant clause (e.g., the negative clauses in $\mathcal{C}$ may be used).

Let v be a subgoal node in T and v' an ancestor of v in t. We say $T \vdash_{red,v,v'} T'$ if: $\lambda(v)$ is unifiable with the complement of $\lambda(v')$ or the complement of a literal from $\iota(v')$. Let σ be the respective *mgu*. We also have $t' = t$, $\lambda'(w) = \lambda(w)\sigma$ and $\iota'(w) = \iota(w)\sigma$ for each node w in t. μ' is the extension of μ by $\mu'(v) = v'$. We say $T \vdash_{ext,v,C} T'$ if there is a variant $l_1 \vee \ldots \vee l_n$ of a clause $C \in \mathcal{C}$ where $\sim\!\lambda(v)$ and one l_i, $i \in \{1, \ldots, n\}$, are unifiable with *mgu* ρ. Furthermore, t' is obtained from t by attaching new nodes $v_1, \ldots, v_n$ below v. $\lambda'(v_j) = l_j\rho$, $1 \leq j \leq n$, and $\iota'(v_j) = \emptyset$ for all $j \in \{1, \ldots, n\}$. For nodes v of t we have $\lambda'(v) = \lambda(v)\rho$ and $\iota'(v) = \iota(v)\rho$. μ' extends μ by $\mu'(v_i) = v$. Factorization ([8,7]) allows the re-use of a solution of $s' = \lambda(v')$ of a node v' for solving the subgoal $s = \lambda(v)$. A factorization step ($\vdash_{fac,v,v'}$) is possible if s and s' can be unified and all inferences that can be used to solve s' are also possible to solve s (see [7] for details). Folding-up is a pessimistic factorization variant (see [7]). Let v be a non-leaf node of T which is head of a closed subtableau. Then, let v' be the deepest ancestor node of v used for reduction into the subtableau with head v or let v' be the root node if such ancestor nodes do not exist. We say $T \vdash_{fu,v} T'$ if $t' = t$, $\lambda' = \lambda$. Moreover, $\iota'(w) = \iota(w)$ for all nodes $w \neq v'$ and $\iota'(v') = \iota(v') \cup \{\sim\lambda(v)\}$.

Search space reduction techniques exclude certain parts from the search space. A search space reduction technique is a predicate on marked connection tableaux. We consider the three techniques *regularity* (*reg*), *tautology-freeness* (*taut*), and *subsumption-freeness* (*subs*) which are used, e.g. in the prover SETHEO. Let $T = ((t, \lambda, \iota), \mu)$ be a marked tableau. $reg(T)$ holds if T is regular, i.e. no nodes on a branch have the same λ-labels. $taut(T)$ holds if no tableau clause contains two complementary literals. Subsumption-freeness can be formulated as follows. Let $>$ be a total and acyclic ordering on the clauses to be refuted. Then, $subs(T)$ holds if T does not contain a tableau clause which is an instance of an input clause C_1 and which is subsumed by another input clause $C_2 \neq C_1$ with $C_1 > C_2$.

The order in which open subgoals are used for inferences is controlled by a *subgoal selection function* which assigns to each open marked tableau a subgoal.

Let $\mathcal{C}$ be a clause set and $\mathcal{S}$ be a set of start clauses for $\mathcal{C}$. Let $calc = (\mathcal{I}, \Sigma, \mathcal{R})$ be a CTC where $\mathcal{I}$ is a set of inference rules, Σ a subgoal selection function, and $\mathcal{R}$ a set of search space reduction techniques. We say $T \vdash_{calc,\mathcal{C},\mathcal{S}} T'$ for two marked tableaux T and T' if T is the trivial tableau and $T \vdash_{start,S} T'$ for $S \in \mathcal{S}$. If T is not trivial, we have either $T \vdash_{fu,v} T'$ for a node v of T, $T \vdash_{I,\Sigma(T),v'} T'$ for $I \in \{red, fac\}$ and a node v' of T, or $T \vdash_{ext,\Sigma(T),C} T'$ for $C \in \mathcal{C}$. In all cases $cond(T')$ must hold for each $cond \in \mathcal{R}$. The search space is a search tree. A *search tree* $\mathcal{T} = \mathcal{T}_{calc,\mathcal{C},\mathcal{S}}$ is a labeled tree, whose root is labeled with the trivial marked tableau. Every node in $\mathcal{T}$ labeled with the marked tableau T has as immediate successors the maximal set of nodes $\{v_1, \ldots, v_n\}$, where each v_i is labeled with a different marked tableau T_i and $T \vdash_{calc,\mathcal{C},\mathcal{S}} T_i$, $1 \leq i \leq n$.

2.3 Proof Search

In order to enumerate a closed tableau for $\mathcal{C}$, i.e. a *proof* of $\mathcal{C}$, normally *iterative deepening search* with backtracking is employed. Iteratively larger finite initial parts of the search tree are explored in depth-first search. The finite segments are defined by so-called *completeness bounds* (see [6,14]). A completeness bound is a function which maps a tableau to a natural number. Furthermore, the following properties must be fulfilled. Let $\mathcal{T} = \mathcal{T}_{calc,\mathcal{C},\mathcal{S}}$. Then, for each $n \in I\!N$ there is only a finite number of tableaux in the search tree which are mapped by $\mathcal{B}$ to a value smaller than or equal to n and $T \vdash_{calc,\mathcal{C},\mathcal{S}} T'$ implies that $\mathcal{B}(T) \leq \mathcal{B}(T')$.

$\mathcal{T}^{\mathcal{B},n}$ is the initial segment of $\mathcal{T}$ which contains all tableaux which are mapped by $\mathcal{B}$ to a value smaller than or equal to the *resource* n. Iterative deepening using $\mathcal{B}$ starts with resource $n \in I\!N$ and iteratively increases n until a proof is found within the (increasing) finite tree $\mathcal{T}^{\mathcal{B},n}$. $\mathcal{B}$ is *label independent* if $\mathcal{B}(T) \leq \mathcal{B}(T')$ for marked tableaux T and T' where T is embedded in T'. Many prominent bounds like the *depth* and *inference bound* ([14,7]) are label independent.

3 Relevancy-Based Clause Selection with Abstractions

At first we will introduce some notions of relevancy. Then, we deal with an abstraction-based framework for selecting relevant clauses.

3.1 Notions of Relevancy

We want to refute a clause set $\mathcal{C}$. Furthermore, let $\mathcal{L} \subseteq \mathcal{C}$ and $\mathcal{M} = \mathcal{C} \backslash \mathcal{L}$. We call $\mathcal{L}$ the *test set*. Our task is to delete irrelevant clauses from $\mathcal{L}$ resulting in a set $\mathcal{L}'$. Then instead of $\mathcal{M} \cup \mathcal{L}$ the set $\mathcal{M} \cup \mathcal{L}'$ is used. $\mathcal{L}$ is a set which may contain a lot of irrelevant clauses. The set $\mathcal{M}$ is considered a set whose relevance is not doubted. E.g., $\mathcal{M}$ is the given theory and $\mathcal{L}$ a set of derivable lemmas. Naturally, also $\mathcal{L} = \mathcal{C}$ and $\mathcal{M} = \emptyset$ may be used.

An intuitive notion of relevancy of a clause set $\mathcal{L}' \subseteq \mathcal{L}$ is "the clauses from $\mathcal{L}'$ are sufficient in order to obtain a proof for $\mathcal{M} \cup \mathcal{L}$ together with some clauses from $\mathcal{M}$". This notion of relevancy, however, ignores the problem that a certain proof has to be *found* in an acceptable amount of time. Thus, it is necessary to include the specific search method in order to estimate the relevancy of clauses.

Our notion of relevancy considers the completeness bound which is used in the final proof run. In each iterative deepening level defined by a resource n we aim at selecting clauses from $\mathcal{L}$ which can provide a refutation of $\mathcal{C}$ with resource n. Furthermore, a relevant clause set should be in some sense minimal.

Definition 1 (Relevant Subset). *Let $\mathcal{C}$ be a clause set, $\mathcal{S} \subseteq \mathcal{C}$ a set of start clauses for $\mathcal{C}$, and $\mathcal{L} \subseteq \mathcal{C}$. Let $\mathcal{M} = \mathcal{C} \backslash \mathcal{L}$. Let calc $= (\mathcal{I}, \Sigma, \mathcal{R})$ be a CTC. $\mathcal{L}' \subseteq \mathcal{L}$, $\mathcal{L}' \neq \emptyset$, is a* relevant subset *of $\mathcal{L}$ for refuting $\mathcal{C}$ w.r.t. calc, $\mathcal{S}$, a completeness bound $\mathcal{B}$, and resource $n \in I\!N$ if in $\mathcal{T} = (\mathcal{T}_{calc,\mathcal{C},\mathcal{S}})^{\mathcal{B},n}$ there is a closed marked tableau T and $\mathcal{L}'$ is the minimal subset of $\mathcal{L}$ which contains all clauses from $\mathcal{L}$ which are needed to infer T. If there is no closed marked tableau in $\mathcal{T}$ then $\mathcal{L}' = \emptyset$ is the only relevant subset of $\mathcal{L}$.*

In each iterative deepening level only a relevant subset of $\mathcal{L}$ should be used in addition to $\mathcal{M}$. This normally results in large savings of unnecessary inferences although we ignore the order in which tableaux are enumerated. Also a small subset of $\mathcal{L}$ which contains a relevant subset of $\mathcal{L}$ is often well-suited. This notion of relevancy considers dependencies between clauses of $\mathcal{L}$. We weaken the notion of relevancy by considering single clauses from $\mathcal{L}$ instead of subsets.

Definition 2 (Relevant Element). *Let $\mathcal{C}$ be a set of clauses, $\mathcal{S} \subseteq \mathcal{C}$ be a set of start clauses for $\mathcal{C}$, and $\mathcal{L} \subseteq \mathcal{C}$. Let calc $= (\mathcal{I}, \Sigma, \mathcal{R})$ be a CTC. We say $l \in \mathcal{L}$ is a* relevant element *of $\mathcal{L}$ for refuting $\mathcal{C}$ w.r.t. calc, S, a completeness bound $\mathcal{B}$, and resource $n \in I\!N$ if there is a relevant subset $\mathcal{L}'$ of $\mathcal{L}$ for refuting $\mathcal{C}$ w.r.t. calc, $\mathcal{S}$, $\mathcal{B}$, and n, which contains l.*

An alternative to our first method is to use during a proof run in each iterative deepening level i only the set $\mathcal{L}^i_{loc} \subseteq \mathcal{L}$ of relevant elements from $\mathcal{L}$ in addition to $\mathcal{M}$. Note that $\mathcal{L}^i_{loc}$ is the union of the relevant subsets of $\mathcal{L}$. Thus, $\mathcal{L}^i_{loc}$ is sufficient to obtain a refutation in the given resource i although it is not of minimal size. Normally, however, the size of $\mathcal{L}^i_{loc}$ is much smaller than that of $\mathcal{L}$.

Both notions of relevancy are *decidable*. A "decision procedure" can be obtained by enumerating all proofs in a given finite segment $(\mathcal{T}_{calc,\mathcal{C},\mathcal{S}})^{\mathcal{B},n}$. Then it has to be examined which clauses from $\mathcal{L}$ appear in closed marked tableaux. But naturally this is not sensible since we are interested in determining the relevancy of clauses from $\mathcal{L}$ without performing a proof search with all clauses from $\mathcal{L}$.

In [3,4] $(\mathcal{T}_{calc,\mathcal{C},\mathcal{S}})^{\mathcal{B},n}$ is only *partially evaluated* and *estimations* of whether certain clauses are relevant are given based on the traversed parts. Here we want to use *abstractions* to provide *exact criteria* for the irrelevance of clauses. The basic idea is to map the search tree of the so-called *ground space* to an abstract search tree in the *abstract space*. By identifying different objects of the ground space we aim at reducing its size. Then, proofs are searched for in the abstract search space and information obtained in these proofs is used to judge whether certain clauses are relevant for finding proofs in the ground space.

3.2 Abstraction-Based Clause Selection

An abstraction is intended to map a given search tree to another search tree.

Definition 3 (Abstraction). *An* abstraction *Φ is a 5-tuple $(\phi_c, \phi_s, \phi_i, \phi_g, \phi_r)$ of abstraction functions. ϕ_c and ϕ_s map a clause set in a signature to a clause set in another signature. We demand that $\phi_s(\mathcal{S}) \subseteq \phi_c(\mathcal{C})$ for all clause sets $\mathcal{S}, \mathcal{C}$ with $\mathcal{S} \subseteq \mathcal{C}$. ϕ_i maps a set of inference rules to another set of inference rules, ϕ_g maps a subgoal selection function to another subgoal selection function, and ϕ_r maps a set of search space reduction techniques to another set of reduction techniques. Let $calc = (\mathcal{I}, \Sigma, \mathcal{R})$ be a CTC. The* abstract connection tableau calculus *$calc_\Phi$ is $(\phi_i(\mathcal{I}), \phi_g(\Sigma), \phi_r(\mathcal{R}))$. Let $\mathcal{C}$ be a set of input clauses and let $\mathcal{S} \subseteq \mathcal{C}$ the set of start clauses for $\mathcal{C}$. Let $\mathcal{T} = \mathcal{T}_{calc,\mathcal{C},\mathcal{S}}$ be the search tree for $\mathcal{C}$ and $\mathcal{S}$. The* abstract search tree *$\mathcal{T}^\Phi$ is given by $\mathcal{T}_{calc_\Phi,\phi_c(\mathcal{C}),\phi_s(\mathcal{S})}$.*

The Use of Abstractions for Relevancy Testing. We introduce general principles regarding the use of abstractions for a relevancy-based clause selection. We are going to define important properties of abstractions which have to be fulfilled in order to provide a relevancy testing. In earlier work on abstractions (cp. [5]) a proof preservation property has been needed. Abstractions have been defined in such a manner that there is still a proof of an abstraction of an inconsistent set of clauses in the abstract search space. However, the property of preserving proofs alone is too weak in order to support a relevancy testing for clauses. The following properties are needed.

Definition 4 (Global, Local Abstraction). *Let $calc = (\mathcal{I}, \Sigma, \mathcal{R})$ be a CTC. Let $\mathcal{B}$ be a completeness bound. Let $\Phi = (\phi_c, \phi_s, \phi_i, \phi_g, \phi_r)$ be an abstraction. Φ is a* global (local) abstraction *w.r.t. calc and $\mathcal{B}$ if the following holds for all inconsistent clause sets $\mathcal{C}$ with start clauses $\mathcal{S}$, and each $m \geq 1$ ($m = 1$). Let $\mathcal{T} = \mathcal{T}_{calc,\mathcal{C},\mathcal{S}}$ be the search tree. Let $T = ((t, \lambda, \iota), \mu)$ be a closed marked tableau in $\mathcal{T}^{\mathcal{B},n}$ for a resource n. Let $C_1, \ldots, C_m$ be clauses from $\mathcal{C}$. Let $p_1, \ldots, p_m$ be positions in T such that $C_i \trianglelefteq T[p_i]$, $1 \leq i \leq m$. Then, a closed marked tableau $T^\Phi = ((t^\Phi, \lambda^\Phi, \iota^\Phi), \mu^\Phi)$ in $(\mathcal{T}^\Phi)^{\mathcal{B},n}$, positions $q_1, \ldots, q_m$ in T^Φ, and clauses $C_i^\Phi \in \phi_c(\{C_i\})$ exist with $C_i^\Phi \trianglelefteq T^\Phi[q_i]$, $1 \leq i \leq m$.*

The basic idea of these notions is that an abstraction should not only guarantee that an inconsistent set of clauses remains inconsistent in the abstract space but also that for each clause 'occurring' in a proof an abstraction exists which 'occurs' in an abstract proof. Local abstractions can be used for selecting single clauses based on our notion of relevancy. In order to support clause selection based on the notion of relevancy of clause sets global abstractions can be used.

Procedure 1 (Input Clause Selection)
Input: clause set $\mathcal{C}$, start clauses $\mathcal{S} \subseteq \mathcal{C}$, test set $\mathcal{L} \subseteq \mathcal{C}$, CTC $calc = (\mathcal{I}, \Sigma, \mathcal{R})$, abstraction $\Phi = (\phi_c, \phi_s, \phi_i, \phi_g, \phi_r)$, completeness bound $\mathcal{B}$, resource $n \in I\!N$
Output: $\mathcal{L}' \subseteq \mathcal{L}$ and $R \subseteq 2^{\mathcal{L}}$

1. Let $\mathcal{T} = \mathcal{T}_{calc,\mathcal{C},\mathcal{S}}$ be the search tree for $calc$, $\mathcal{C}$, and $\mathcal{S}$.
2. Enumerate all closed marked tableaux $T_1, \ldots, T_k$ in $(\mathcal{T}^{\Phi})^{\mathcal{B},n}$. For all $i \in \{1, \ldots, k\}$ let $T_i = ((t_i, \lambda_i, \iota_i), \mu_i)$.
3. $\mathcal{L}' = \{C \in \mathcal{L} : \exists i \in \{1, \ldots, k\}, p \in Pos(t_i), C^{\Phi} \in \phi_c(\{C\})$ with $C^{\Phi} \trianglelefteq T_i[p]\}$
4. $R = \{\{C_1, \ldots, C_l\} \subseteq \mathcal{L} : \exists i \in \{1, \ldots, k\}$ and pairwise distinct $p_1, \ldots, p_l \in Pos(t_i)$ such that $\forall j \in \{1, \ldots, l\}\ \exists C_j^{\Phi} \in \phi_c(\{C_j\})$ with $C_j^{\Phi} \trianglelefteq T_i[p_j] \wedge \nexists p \in Pos(t_i)$, $p \neq p_m$ $(1 \leq m \leq l)$, $C \in \mathcal{L}$, $C^{\Phi} \in \phi_c(\{C\})$ with $C^{\Phi} \trianglelefteq T_i[p]\}$.

Theorem 1 ([2]). *Let $\mathcal{C}$ be a set of clauses. Let $\mathcal{S} \subseteq \mathcal{C}$ be a set of start clauses. Let $\mathcal{L} \subseteq \mathcal{C}$. Let calc $= (\mathcal{I}, \Sigma, \mathcal{R})$. Let $\mathcal{T} = \mathcal{T}_{calc,\mathcal{C},\mathcal{S}}$ be the search tree for $\mathcal{C}$ and $\mathcal{S}$. Let $\mathcal{B}$ be a completeness bound and $n \in I\!N$ be a resource. Let Φ be global (local) w.r.t. calc and $\mathcal{B}$. Let $R \subseteq 2^{\mathcal{L}}$ ($\mathcal{L}' \subseteq \mathcal{L}$) be the output of Procedure 1 applied to $\mathcal{C}$, $\mathcal{S}$, $\mathcal{L}$, calc, Φ, $\mathcal{B}$, and n. Let $\mathcal{O} \in 2^{\mathcal{L}} \backslash R$ ($C \in \mathcal{L} \backslash \mathcal{L}'$). Then, $\mathcal{O}$ (C) is not a relevant subset (element) of $\mathcal{L}$ for refuting $\mathcal{C}$ w.r.t. calc, $\mathcal{S}, \mathcal{B}$, and n.*

We can use local abstractions in order to discard certain clauses from the test set $\mathcal{L}$ which are definitely useless in the given resource n. Thus, we can obtain a subset of $\mathcal{L}$ which is a superset of the set of all relevant elements w.r.t. n. This set may be used for a refutation run within the given resource instead of $\mathcal{L}$.

Global abstractions can further reduce the number of clauses to be used in a proof run. We may employ one or many clause sets from the output R of Procedure 1 instead of $\mathcal{L}$ which can provide large search reductions. In practice this can be done by partitioning the set R in l subsets $R_1, \ldots, R_l$ and starting l provers $P_1, \ldots, P_l$ where each prover P_i uses the union of the clause sets of R_i instead of $\mathcal{L}$. These provers can run in parallel or in succession (in the latter case with small time-outs). Thus, a further gain of efficiency compared to the use of $\mathcal{L}'$ may be possible since we work with smaller clause sets. The globality of abstractions is hence desirable. Obviously, global abstractions are also local. The inverse relation need not hold as we will see now.

Prominent Abstractions. Henceforth, we consider two classes of abstractions: *generalization techniques* as introduced in [11] and so-called *symbol abstractions* (see e.g. [10,5]). This study clarifies the relationship between the locality and the globality property as introduced before. We also get some insights into the way how 'concrete' abstractions must look like in order to fulfill our 'abstract' properties of abstractions.

An abstraction $\Phi = (\phi_c, \phi_s, \phi_i, \phi_g, \phi_r)$ is a *generalization abstraction* if the abstraction functions ϕ_c and ϕ_s look as follows. $\phi_c(\mathcal{C})$ for a clause set $\mathcal{C} = \{C_1, \ldots, C_n\}$ equals the set which is obtained from $\bigcup_{i=1}^{n} \phi'_c(C_i)$ after identifying permutation variants of clauses. $\phi'_c(C)$ for a clause C is given by a non empty clause set $\mathcal{D}$ with $D \trianglelefteq C$ for all $D \in \mathcal{D}$. $\phi_s(\mathcal{S})$ is given by a subset of $\phi_c(\mathcal{S})$. This restriction prevents an uncontrolled change of the size of the abstract search tree compared to the ground search tree.

We consider variants of generalization abstractions. Φ is a *literal deleting generalization abstraction* if ϕ'_c maps a clause C to a clause set $\{D\}$ such that $D \triangleleft C$. Furthermore, we have $\phi_s = \phi_c$. Other techniques based on generalizations do not delete literals but split a clause in simpler subsuming clauses. The *joint subsumption abstraction* of Plaisted ([12]) maps a clause $C = l_1 \vee \ldots \vee l_k$ to a set T in such a manner that a set of clauses $\{D_1, \ldots, D_n\}$ exists where the union of the literals occurring in $\{D_1, \ldots, D_n\}$ equals $\{l_1, \ldots, l_k\}$ and for each clause D_i, $1 \leq i \leq n$, there is $E \in T$ and σ with $D_i = E\sigma$. Furthermore, $\phi_s = \phi_c$. A variant of such abstractions is defined in [2]. *Splitting abstractions* basically realize the splitting of the *tail literals* of all so-called *contrapositives* of input clauses (see [14]) instead of only the clauses. E.g., the clause $p \vee q \vee r$ is splitted in $\{p \vee q, p \vee r, q \vee r\}$ instead of $\{p \vee q, r\}$ if we want to split ground clauses in two-literal abstract clauses (see [2]).

Theorem 2 ([2]).

1. *There are a CTC calc, a label independent completeness bound $\mathcal{B}$, and a literal deleting abstraction (joint subsumption abstraction) $\Phi = (\phi_c, \phi_i, \phi_g, \phi_r)$ with $\mathcal{I} \subseteq \phi_i(\mathcal{I})$ and $\phi_r(\mathcal{R}) = \emptyset$ such that Φ is not local w.r.t. calc and $\mathcal{B}$.*
2. *Let $\Phi = (\phi_c, \phi_s, \phi_i, \phi_g, \phi_r)$ be a splitting abstraction. Let calc $= (\mathcal{I}, \Sigma, \mathcal{R})$. Let $fu, fac \notin \mathcal{I}$. Let $\mathcal{B}$ be a completeness bound which is label independent. Let $\mathcal{I} \subseteq \phi_i(\mathcal{I})$, $\phi_r(\mathcal{R}) \subseteq \mathcal{R}$. Let subs $\notin \phi_r(\mathcal{R})$. Then, Φ is local w.r.t. calc and $\mathcal{B}$. But in general Φ is not global w.r.t. calc and $\mathcal{B}$.*
3. *Let calc $= (\mathcal{I}, \Sigma, \mathcal{R})$. Let $fu \in \mathcal{I}$ or $fac \in \mathcal{I}$. Then there are a label independent bound $\mathcal{B}$ and a splitting abstraction $\Phi = (\phi_c, \phi_s, \phi_i, \phi_g, \phi_r)$ with $\mathcal{I} \subseteq \phi_i(\mathcal{I})$ and $\phi_r(\mathcal{R}) = \emptyset$ such that Φ is not local w.r.t. calc and $\mathcal{B}$.*

As one can expect literal deleting abstractions cannot be used for relevancy testing since the deletion of literals excludes certain subdeductions from abstract proofs. But also the prominent joint subsumption technique cannot be used for a relevancy testing although no literals are deleted. The abstraction is even not local if we do not restrict the start rule in the abstract search space and use $\phi_s(\mathcal{S}) = \phi_c(\mathcal{C})$ (cp. [2]). Splitting abstractions are local if we do not use factorization techniques. The splitting of ground deductions into several abstract deductions which is performed by splitting contrapositives, however, makes a reuse of solutions of subgoals, as required by factorization, impossible (see [2]). Analogously, the splitting of deductions damages the globality property.

Since the use of factorization is desirable we want to consider two abstraction types in more detail which do not change the size of clauses and preserve the structure of proofs. We introduce another type of generalization abstraction, namely *clause size invariant* generalization abstractions. Furthermore, we deal with *symbol abstractions*.

Φ is a *clause size invariant generalization abstraction* if ϕ'_c maps a clause $p_1(t^1_1, \ldots, t^1_{n_1}) \vee \ldots \vee p_m(t^m_1, \ldots, t^m_{n_m})$ to a subsuming clause $p_1(s^1_1, \ldots, s^1_{n_1}) \vee \ldots \vee p_m(s^m_1, \ldots, s^m_{n_m})$ and $\phi_s = \phi_c$. An example is a function $\phi'_c = \phi^{msg}_c$ which maps a clause C to its *most special generalizer* whose maximal literal depth is limited to a given value $d \in I\!N$. E.g., for $d = 2$ we have $\phi^{msg}_c(p(a, g(b)) \vee p(g(b), g(a))) = p(a, X) \vee p(X, Y)$.

Symbol abstractions are induced by the specific function ϕ_c. We have $\phi_s = \phi_c$. ϕ_c defines equivalence classes on clauses by identifying some predicate or function symbols. Let $\sim_{\mathcal{P}}$ be a relation on predicate symbols of the same arity. Analogously $\sim_{\mathcal{F}}$ defines equivalence classes on the function symbols $\mathcal{F}$ of the same arity. $[p]_{\sim_{\mathcal{P}}}$ denotes $\{q : q \sim_{\mathcal{P}} p\}$. Analogously we use $[f]_{\sim_{\mathcal{F}}}$. Terms over $\mathcal{F}$ and variables $\mathcal{V}$ are mapped as follows using a function ϕ'_c on terms. $\phi'_c(t) = X$ if $t = X \in \mathcal{V}$ and $\phi'_c(t) = [f]_{\sim_{\mathcal{F}}}(\phi'_c(t_1), \ldots, \phi'_c(t_n))$ if $t = f(t_1, \ldots, t_n)$. ϕ'_c is extended to atoms, literals, and clauses in a natural way. $\phi_c(\mathcal{C})$ of a clause set $\mathcal{C}$ is obtained from the set of the abstractions of its elements by identifying permutation variants. Technically, we introduce for each equivalence class a new symbol and thus the clauses are mapped to a new signature. E.g., let $\mathcal{P} = \{p\}$ and $\mathcal{F} = \{a, b, c\}$. Let $\sim_{\mathcal{P}} = \emptyset$ and $a \sim_{\mathcal{F}} b$. We introduce a new symbol $\cdot_{a,b}$ for the class $\{a, b\}$ and have $\phi'_c(p(a)) = \phi'_c(p(b)) = p(\cdot_{a,b})$ and $\phi'_c(p(c)) = p(c)$.

Since these two abstraction types can maintain the structure of a tableau we use a notion of an abstraction of a tableau (defined in [2]). Intuitively a tableau T^{Φ} is an *abstraction of a tableau* T of the ground search tree if T^{Φ} is created by an inference chain involving abstract clauses which completely corresponds to the inferences performed to create T. E.g., if for each $C = l_1 \vee \ldots \vee l_k \in \mathcal{C}$ the clause $C' = \phi'_c(C) = l'_1 \vee \ldots \vee l'_k$ is part of $\phi_c(\mathcal{C})$, then all inferences to infer T are also performed in the same order and to the same positions in order to infer T^{Φ}. The only difference is that the use of a clause C is replaced by the use of C'. In the case where only a permutation variant of $\phi'_c(C)$ is in $\phi_c(\mathcal{C})$ inferences have to be performed to 'corresponding' positions (details can be found in [2]).

While there is at most one abstraction of a tableau in the abstract search tree many tableaux of the ground space may be abstracted to one tableau which reduces the branching rate of the search space. We should note, however, that the subgoal selection function which is used in the abstract proof run can make it impossible to perform the inferences needed to infer an abstraction of a ground proof. This is no problem if folding-up is not used as Theorem 3 states.

Theorem 3 ([2]).

1. *Let calc* $= (\mathcal{I}, \Sigma, \mathcal{R})$. *Let* $fu \notin \mathcal{I}$. *Let* $\Phi = (\phi_c, \phi_s, \phi_i, \phi_g, \phi_r)$ *be a clause size invariant generalization (symbol) abstraction. Let* $\mathcal{B}$ *be a completeness bound which is label independent. Let* $\mathcal{I} \subseteq \phi_i(\mathcal{I})$. *Let* $\phi_r(\mathcal{R}) \subseteq \mathcal{R}$ *and* $subs \notin \phi_r(\mathcal{R})$ $(\phi_r(\mathcal{R}) = \emptyset)$. *Then,* Φ *is global w.r.t. calc and* $\mathcal{B}$.
2. *There are calc* $= (\mathcal{I}, \Sigma, \mathcal{R})$ *with* $fu \in \mathcal{I}$, *a clause size invariant generalization (symbol) abstraction* $\Phi = (\phi_c, \phi_s, \phi_i, \phi_g, \phi_r)$ *with* $\phi_i(\mathcal{I}) \supseteq \mathcal{I}$, $\phi_r(\mathcal{R}) = \emptyset$, *and a label independent bound* $\mathcal{B}$ *such that* Φ *is not local w.r.t. calc and* $\mathcal{B}$.

Thus, if we do not use folding-up (and take care when using search space reduction techniques) we can guarantee the globality property. Specifically, we can derive tableaux which are *equivalent* (modulo renaming variables) to abstractions of each ground proof in the abstract space (see [2]).

A simultaneous use of folding-up and a dynamic choice of subgoals is in general not assisted. But when "associated" subgoals are chosen in certain marked tableaux of the ground search space (which are on a path to a proof in the search tree) and their (isomorphic) abstractions in the abstract search space, at least an abstraction of *one* closed tableau can be derived (see [2]). Then, Procedure 1 can provide the selection of a superset of a relevant subset of $\mathcal{L}$. In order to avoid the deletion of relevant elements from $\mathcal{L}$ the conditions must be even more restrictive (see [2]). Specifically, this may even be impossible if the subgoal selection function of the *ground space* does not show a certain behavior.

However, it is normally sufficient to select clauses occurring in one specific proof. Nevertheless, it is unclear which subgoal selection function should be used in the abstract space. When using clause size invariant generalization abstractions, the same function as in the ground space is normally no good choice. Often weighting functions are used which judge the complexity of a literal regarding its syntactical structure (see e.g. [9]) and choose the most complex subgoal. Then, the order in which subgoals are tackled is usually different between tableaux of the ground space and their abstractions. If we employ symbol abstractions and weighting functions on literals for subgoal selection in the ground and abstract space we can often derive an abstraction of a ground proof. This is because the computation of the literal weight is not influenced by the abstraction. The only problem is that conflicts between equally weighted subgoals may differently be resolved in the ground and the abstract space. But it is likely that at least the abstraction of one proof can be derived. Due to this advantage we have used symbol abstractions in the following case study.

4 Case Study: Symbol Abstractions

In order to make the application of symbol abstractions viable we consider some new aspects concerning abstraction generation and clause selection.

4.1 Automatic Generation of Symbol Abstractions

Basically, in order to use an abstraction for clause selection the abstract search tree should be considerably smaller than the original search tree. Otherwise, a search in the abstract search tree offers no gain of efficiency. Moreover, the abstraction should be "exact" enough in order to exclude abstractions of many ground clauses from proofs of the abstract search space. Otherwise too many irrelevant clauses may be maintained when selecting clauses with Procedure 1.

In [2] it is shown that under certain conditions (see above) for each closed marked tableau $T = ((t, \lambda, \iota), \mu)$ of the ground search space a tableau $T^{\Phi} = ((t^{\Phi}, \lambda^{\Phi}, \iota^{\Phi}), \mu^{\Phi})$ (equivalent to the abstraction of T) can be derived where for each node v of t holds $\phi'_c(\lambda(v)) = (\lambda^{\Phi}(\pi(v)))\mu$ for a variable renaming μ and an isomorphism π (see [2]). Using this property we refine our two general claims on sensible abstractions and demand the following for the *construction of symbol abstractions*. The size of the abstract input clause set should be smaller than that of the original input clause set. Then, the size of the search tree may be decreased. Furthermore, there should be only few abstract proofs T' which are not equivalent to abstractions of ground proofs. Such marked tableaux T' are called *false proofs* and are responsible for the selection of unnecessary clauses. We may also select a high number of clauses from $\mathcal{L}$ although we have only few false proofs. This can happen if many clauses from $\mathcal{L}$ are mapped to an equal abstract clause which occurs in an abstract proof. Thus, different clauses from $\mathcal{L}$ should be identified very carefully. All in all, three different aspects are important: a *reduction* of the number of input clauses, a *prevention of false proofs*, and the ability of *distinction* between the clauses whose relevance should be investigated.

We merely sketch our algorithm for abstraction generation (see [2] for details). A sensible choice of ϕ_g, ϕ_r, and ϕ_i is no problem. $\phi_r(\mathcal{R}) = \emptyset$ should be chosen. Furthermore, $\phi_i(\mathcal{I}) = \mathcal{I}$ and $\phi_g(\Sigma) = \Sigma$ is sensible. Thus, in the following we are only looking for the optimal choice of ϕ_c for a set $\mathcal{C}$ of input clauses.

Basically, we aim at constructing the relations $\sim_{\mathcal{P}}$ and $\sim_{\mathcal{F}}$ with a quality function *qual* where a small value indicates a high quality of the induced abstraction. *qual* is a linear combination of three quality functions $qual_{red}$, $qual_{fp}$, and $qual_{dist}$ which judge the reduction, false proof prevention, and distinction capabilities of a symbol abstraction, respectively. $qual_{red}(\phi_c)$ is simply given as the size of the set $\phi_c(\mathcal{C})$. The false proof prevention capability $qual_{fp}(\phi_c)$ counts how many different symbols are identified by the abstraction (see [2]). The distinction capability $qual_{dist}$ computes the size of the maximal subset of $\mathcal{C}$ where all clauses in the subset are mapped by ϕ_c to permutation variants of each other.

Now we want to construct abstractions with an optimal quality value. An explicit construction of all abstractions is not sensible due to complexity reasons (see [2]). Thus, we use an agglomerative approach which successively creates $\sim_{\mathcal{P}}$ and $\sim_{\mathcal{F}}$ by identifying certain symbols with the same arity. We start with $\sim_{\mathcal{P}}$ and $\sim_{\mathcal{F}}$ such that $[p]_{\sim_{\mathcal{P}}} = \{p\}$ and $[f]_{\sim_{\mathcal{P}}} = \{f\}$ for each predicate or function symbol p or f, respectively. Then, successively $\sim_{\mathcal{P}}$ or $\sim_{\mathcal{F}}$ is changed by 'merging' the two equivalence classes such that the resulting abstraction is the best one obtainable by merging two equivalence classes.

The quality of the abstraction is judged using $qual_{red}$, $qual_{fp}$, and $qual_{dist}$. Implicitly we employ $qual_{fp}$ due to the agglomerative approach. $qual_{dist}$ is used without any change. But we slightly change $qual_{red}$. Merging two equivalence classes alone may not lead to an identification of input clauses and the need for the identification of certain symbols can only be recognized after identifying further symbols. Thus, we perform some 'preview' and look whether 'structural distances' between clauses (which can be identified by identifying symbols) are reduced by merging two classes (see [2]).

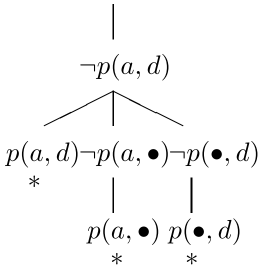
The algorithm stops if a maximal size of an equivalence class w.r.t. $\sim_{\mathcal{P}}$ or $\sim_{\mathcal{F}}$ (maximal *cluster size*) is reached. Using this algorithm we can often approximate the minimization of *qual* in a rather well-suited manner.

4.2 Improved Clause Selection Using Partial Mapping Back

Now, we strengthen the method of Procedure 1. So called *partial mapping-back* methods reduce the number of selected clauses. We try to replace some tableau clauses of an abstract proof by clauses of the ground space. For simplicity reasons we only consider calculi where no factorization techniques are used.

Definition 5 (Partial Mapping Back). *Let $T^{\Phi} = ((t^{\Phi}, \lambda^{\Phi}, \iota^{\Phi}), \mu^{\Phi})$ be a proof in the abstract tree $\mathcal{T}^{\Phi}$. Let $\mathcal{E} = (E_1, \ldots, E_n)$ be a list of clauses where each E_i is part of the test set $\mathcal{L} \subseteq \mathcal{C}$. Let $(p_1, \ldots, p_n)$ be a list of positions in t^{Φ} such that $\phi'_c(E_i) \trianglelefteq T^{\Phi}[p_i]$. Let π be a mapping where $\pi(p_i)$ is a permutation variant of E_i. Let $k \in I\!N$ and $k' = \min(\{k, |Pos(t^{\Phi})| - n\})$. We say T^{Φ}, $\mathcal{E}$, and $(p_1, \ldots, p_n)$ can be* partially mapped back with degree *k if for all pairwise distinct $q_1, \ldots, q_{k'} \in Pos(t^{\Phi}) \backslash \{p_1, \ldots, p_n\}$ there is an extension of π with $\pi(q_i)$ is a permutation variant of an element from $\mathcal{C}$, and a substitution ρ with: $T^{\Phi}[p_i] = \phi'_c(\pi(p_i)\rho)$, $1 \le i \le n$, $T^{\Phi}[q_i] = \phi'_c(\pi(q_i)\rho)$, $1 \le i \le k'$, and for all $p, q \in \{p_1, \ldots, p_n, q_1, \ldots, q_{k'}\}$ where $t^{\Phi}|(p.i)$, $i \in I\!N$, is marked with $t^{\Phi}|(q.j)$, $j \in I\!N$, it holds that $(\pi(p)\rho, i)$ is complementary to $(\pi(q)\rho, j)$.*

Example 1. Let $\mathcal{C} = \{\neg p(X,Y), p(X,Y) \vee \neg p(X,Z) \vee \neg p(Z,Y), p(a,b), p(b,d), p(c,d)\}$. Let $\mathcal{L} = \{p(b,d), p(c,d)\}$. Let $b \sim_{\mathcal{F}} c$. The following abstract proof $T^{\Phi} = ((t^{\Phi}, \lambda^{\Phi}, \iota^{\Phi}), \mu^{\Phi})$ exists where $[b]_{\sim_{\mathcal{F}}}$ is denoted by the new symbol $\bullet$.



Let $\mathcal{E}_1 = (p(c,d))$ and let $\mathcal{E}_2 = (p(b,d))$. Let $p_1 = p_2 = (1.3)$. Instances of the abstractions of $p(c,d)$ and $p(b,d)$ occur at the positions p_1 and p_2. T^{Φ}, $\mathcal{E}_2$, and (p_2) can be partially mapped back with degree 2. But T^{Φ}, $\mathcal{E}_1$, and (p_1) cannot be partially mapped back with degree 2. For $q_1 = (1)$ and $q_2 = (1.2)$ there are no permutation variants C_1 and C_2 of clauses from $\mathcal{C}$, and no substitution σ with: $\phi'_c(C_1\sigma)$ and $\phi'_c(C_2\sigma)$ equal $T^{\Phi}[q_1]$ and $T^{\Phi}[q_2]$, respectively, $\phi'_c(p(c,d)\sigma) = T^{\Phi}[p_1]$, and $(C_1\sigma, 2)$ and $(C_2\sigma, 1)$ as well as $(C_1\sigma, 3)$ and $(p(c,d)\sigma, 1)$ are complementary. Thus, it can be detected that no connection tableau of the ground space which contains $p(c,d)$ at position p_1 can be abstracted to T^{Φ}.

The relevancy estimation can be improved when employing such partial mapping back mechanisms. We can easily define an extension of Procedure 1 where the result sets $\mathcal{L}'$ or R contain only clauses or clause sets whose abstractions occur in proofs T^{Φ} such that a partial mapping back exists for T^{Φ}, the considered clauses, and the positions they occur in. Additionally a partial mapping back test can be performed *during* the proof run for a tableau, an empty list of clauses, and an empty list of positions. Then, a high number of false proofs may be prevented and we obtain an appropriate control during the proof run. (This is opposite to approaches which use *hierarchies* of abstract spaces, e.g. [11,1]).

5 Experiments with Lemma Selection

Our new techniques should support a clause selection in large theories. Thus, we have experimented with clause sets from mathematical domains which are augmented with lemmas (see e.g., [13,4]). Since we want to apply symbol abstractions we have chosen test domains which contain many predicate or function symbols. Then, we may reduce the size of the input sets by identifying different symbols. The domains FLD, GEO, NUM, and SET of the library TPTP v.2.0.0 met the above mentioned conditions. Henceforth, we consider abstractions which only identify function symbols. All problems unsolvable with the prover SETHEO using a time-out of 10 seconds on a Sun Ultra 2 constitute our test set.

In order to solve a given problem with our abstraction controlled prover SETHEO/ABS we use the following procedure whose accumulated run time must not exceed T seconds.

At first the calculus and the completeness bound of SETHEO are configured using certain problem characteristics (cp. [9]). Then *unit lemmas* are generated as described in [13], i.e. systematically all lemmas are generated whose proof depth is smaller than 4. We use an upper bound of 800 unit lemmas. The set $\mathcal{C} \cup \mathcal{C}_L$ which consists of the 'old' input clause set $\mathcal{C}$ and the lemma set $\mathcal{C}_L$ forms the new clause set to be refuted. After that the abstraction of the clause set $\mathcal{C} \cup \mathcal{C}_L$ is computed. This step is parameterized with a value N_{cl} for the maximal cluster size. Then, abstract proof runs, clause selection, and proof runs in the ground space are interleaved in a cyclic process. In the first cycle a start resource $n \in I\!N$ is used.

The first step of a cycle is a proof run with the abstraction of $\mathcal{C} \cup \mathcal{C}_L$. We enumerate all closed tableaux in the abstract search tree defined by the resource n until either the segment is completely explored or a time limit T_{abs} is exceeded.

Now, input clauses are selected. If all proofs of the current abstract segment have been enumerated, $\mathcal{L} = \mathcal{C} \cup \mathcal{C}_L$ serves as test set. Let $\mathcal{L}' \subseteq \mathcal{L}$ be the output of Procedure 1 which additionally uses a mapping back value k. The new input clause set for a ground proof attempt is $\mathcal{C}' = \mathcal{L}'$. If the abstract proof run has stopped because the given time limit T_{abs} has been reached, we use as test set $\mathcal{L} = \mathcal{C}_L$. Since it is not guaranteed that an abstraction of a ground proof has been enumerated we maintain all original clauses. Again we select clauses from $\mathcal{L}$ with Procedure 1 resulting in $\mathcal{L}' \subseteq \mathcal{L}$. The set of remaining clauses is $\mathcal{C}' = \mathcal{C} \cup \mathcal{L}'$.

In the last step of the cycle we try to refute the clause set $\mathcal{C}'$ using resource n. If a proof can be found before the global time limit of T seconds is reached we can stop. Otherwise, if we have unsuccessfully enumerated all tableaux in the ground segment, n is increased in an appropriate way (see [14,9]) and the cycle starts again. The exact parameters used in our experiments are given shortly.

We experimented with SETHEO/ABS, a version of SETHEO which does not employ lemmas, and the system SETHEO/A which uses all generated lemmas.

At first we analyze the results obtained in FLD where our method has worked best. SETHEO/ABS has been configured as follows. The time-out T_{abs} for an abstract proof run was $T_{abs} = 10$ seconds. The degree of mapping back was $k = 1$ and the maximal cluster size $N_{cl} = 3$. In Table 1 we depict the run

Table 1. Experiments in the TPTP library

problem	SETHEO	SETHEO/A	SETHEO/ABS	$\|\mathcal{C} \cup \mathcal{C}_L\|$	$\|\phi_c(\mathcal{C} \cup \mathcal{C}_L)\|$	$\|\mathcal{C}'\|$
FLD005-1	—	183 s	23 s	829	54	37^*
FLD008-3	—	5 s	7 s	264	53	64^*
FLD022-3	76 s	1 s	33 s	832	128	262^*
FLD027-3	—	11 s	28 s	831	101	201^*
FLD028-3	107 s	—	151 s	833	104	33
FLD031-1	—	—	225 s	830	218	66
FLD032-1	—	—	98 s	830	216	55^*
FLD041-3	—	—	38 s	830	103	156
FLD060-2	110 s	1 s	22 s	834	129	75^*
FLD066-3	124 s	—	—	112	55	40

FLD	SETHEO	SETHEO/A	SETHEO/ABS	GEO	SETHEO	SETHEO/A	SETHEO/ABS
$\leq$ 10 s	0	18	4	$\leq$ 10 s	0	3	2
$\leq$ 1 min	5	19	23	$\leq$ 1 min	10	5	9
$\leq$ 5 min	14	20	26	$\leq$ 5 min	17	7	17

NUM	SETHEO	SETHEO/A	SETHEO/ABS	SET	SETHEO	SETHEO/A	SETHEO/ABS
$\leq$ 10 s	0	0	0	$\leq$ 10 s	0	9	0
$\leq$ 1 min	0	0	1	$\leq$ 1 min	18	11	15
$\leq$ 5 min	0	0	2	$\leq$ 5 min	36	21	41

times of selected problems (see [2] for a complete list). '—' shows that a problem could not be solved in 300 seconds. Additionally we give the number of clauses in $\mathcal{C} \cup \mathcal{C}_L$, the size of the abstraction of $\mathcal{C} \cup \mathcal{C}_L$, and the number of clauses from $\mathcal{C}'$. $\mathcal{C}'$ contains the clauses which are used in the last ground proof attempt where either the goal is proved or a time-out fail occurred. If this number is marked with '*' all existing abstract proofs of the considered segment have been enumerated.

A significant increase of the performance of SETHEO can be obtained using abstractions. The number of abstract clauses is much smaller than the number of ground clauses. In many cases this leads to a very small segment of the abstract search tree which contains an abstraction of a proof. In many cases we can exhaustively enumerate this segment in 10 seconds, even for non-trivial problems like FLD005-1 and FLD032-1. In the other cases a more detailed analysis shows that the abstract run time is sufficient to enumerate tableaux which are equivalent to abstractions of proofs in the ground space.

Also in the other TPTP domains satisfying results could be obtained. In Table 1 we depict the number of problems which can be solved with SETHEO, SETHEO/A, and SETHEO/ABS. We show the number of problems which could be solved after 10 seconds, 1 minute, and 5 minutes. We used a mapping back degree value $k = 1$. The cluster size was limited to a value of $N_{cl} = 3$ in the FLD, GEO, and SET domain. In NUM a value $N_{cl} = 4$ yields the best results. Again we can see that a use of abstractions increases the efficiency of SETHEO. Only in GEO we could not improve on the conventional system but can at least obtain the same results whereas an unbounded lemma use entails bad results.

In all of our test domains a spare identification of symbols significantly reduces the number of input clauses. This is due to our specific theories because the systematically generated lemmas (see [13,3]) normally have a similar structure such that the identification of function symbols leads to an identification of clauses. Thus, we could automatically derive abstractions which decrease the search space but are still 'exact' enough to exclude many unnecessary clauses.

6 Discussion and Related Work

Abstractions have been widely used in ATP with moderate success (see [15,12,1]). For an overview we refer to [5]. Abstractions have mainly been used to guide the search for proofs of the ground space. Our application of relevancy testing for the CTC has not been investigated in the past.

The closest relationship to our work has the work done by Plaisted (cp. [10,11,12]). Among other techniques generalization and variants of symbol abstractions have been used to guide resolution proofs. In contrast we deal with relevancy testing for the CTC and iterative deepening proof procedures. We have introduced new notions of relevancy and a general framework regarding the use of abstractions for relevancy testing. Furthermore, we have investigated specific problems when using the CTC. Specifically, the use of factorization techniques together with a free subgoal choice turns out problematic. A further novel aspect of our work is the *automatic* generation of abstractions. The newly developed partial mapping back techniques are very promising for future developments. The use of *constraints* which represent allowed bindings for symbols together with fast consistency tests which check a subset of the given constraints may provide efficient abstract proof runs where only few false proofs are generated.

References

1. D. Barker-Plummer. Gazing: An Approach to the Problem of Definition and Lemma Use. *JAR*, 8:311–344, 1992.
2. M. Fuchs. Abstraction-Based Relevancy Testing for Model Elimination. AR-Report AR-98-5, 1998, Technische Universität München, Institut für Informatik, 1998.
3. M. Fuchs. Relevancy-Based Lemma Selection for Model Elimination using Lazy Tableaux Enumeration. In *Proc. ECAI-98*, pages 346–350, 1998.
4. M. Fuchs. System Description: Similarity-Based Lemma Generation for Model Elimination. In *Proceedings of CADE-15*, pages 33–37, 1998.
5. F. Giunchiglia and T. Walsh. A Theory of Abstraction. *AI*, 56(2-3):323–390, 1992.
6. R. Korf. Macro-Operators: A Weak Method for Learning. *AI*, 26:35–77, 1985.
7. R. Letz, K. Mayr, and C. Goller. Controlled Integration of the Cut Rule into Connection Tableau Calculi. *JAR*, 13:297–337, 1994.
8. D.W. Loveland. *Automated Theorem Proving: a Logical Basis.* North-Holland, 1978.
9. M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. The Model Elimination Provers SETHEO and E-SETHEO. *JAR*, 18(2), 1997.
10. D.A. Plaisted. Theorem Proving with Abstraction. *AI*, 16:47–108, 1981.
11. D.A. Plaisted. Abstraction Using Generalization Functions. In *Proceedings of CADE-8*, pages 365–376, 1986.
12. D.A. Plaisted. Mechanical Theorem Proving. In *Formal Techniques in Artificial Intelligence*. Elsevier Science Publisher B.V., 1990.
13. J. Schumann. Delta - a bottom-up preprocessor for top-down theorem provers. system abstract. In *Proceedings of CADE-12*, 1994.
14. M.E. Stickel. A prolog technology theorem prover: Implementation by an extended prolog compiler. *JAR*, 4:353–380, 1988.
15. J.D. Tenenberg. Preserving Consistency across Abstraction Mappings. In *Proceedings of IJCAI-87*, pages 1011–1014, 1987.

A Breadth-First Strategy for Mating Search

Matthew Bishop[⋆]

Department of Mathematical Sciences, Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
mbishop+@cs.cmu.edu

Abstract. Mating search is a very general method for automating proof search; it specifies that one must find a complete mating, without specifying the way in which this is to be achieved. It is the foundation of TPS, an automated theorem-proving system for simply-typed lambda-calculus, and has proven very effective in discovering proofs of higher-order theorems. However, previous implementations of mating search have all relied on essentially the same mating search method: enumerating the paths through a matrix of literals. This is a depth-first strategy which is both computationally expensive and vulnerable to blind alleys in the search space; in addition, the incremental computation of unifiers which is required is, in the higher-order case, very inefficient. We describe a new breadth-first mating search method, called component search, in which matings are constructed by taking unions from a fixed list of smaller matings, whose unifiers are stored and manipulated as directed graphs. Component search is capable of handling much larger search spaces than were possible with path-enumeration search, and has produced fully automatic proofs of a number of interesting theorems which were previously intractable.

1 Introduction

The mating search (connection) method [And81,Bib82] is the foundation of TPS, an automated theorem-proving system for Church's simply-typed λ-calculus [Chu40]. It has proven very effective as a component of a procedure for discovering proofs of higher-order theorems. The salient feature of this method is that it requires consideration of the vertical paths through a two-dimensional matrix of literals. This matrix, also called a *jform*, represents the negation-normal form of the Skolemized negation of the theorem to be proven. Specifically, a proof is represented by a set of pairs of literals (a *mating*) with a substitution that simultaneously makes each of these pairs complementary, and such that every vertical path through the matrix contains at least one of these pairs (i.e. is *spanned by the mating*; a path which is not spanned is said to be *open*). Such a mating is said to be *complete*.

This characterisation of a proof suggests an algorithm for proof search: given such a matrix, begin with the empty mating and an empty substitution. At each

⋆ Supported by the National Science Foundation under grants CCR-9624683 and CCR-9732312, and by Intel Corporation through an equipment grant.

step, given a mating M, find the leftmost open path in the matrix, and attempt to span it by adding a new pair of literals (*connection*) to M and verifying that the result is unifiable. If there are no open paths, then M is complete and we have succeeded. If there is an open path but it cannot be spanned, then backtrack by removing the most recently added connection in M. If we attempt to backtrack when M is empty, then fail.

The above method was the basis of every search procedure in TPS. (Of course, the method as presented is both naïve and incomplete; a full discussion of possible optimizations, or of the modifications required to allow for quantifier duplication and set-variable instantiation, is beyond the scope of this paper. We refer the reader to [And89,ABI$^+$96,Iss90] for more details.) We shall refer to such a search as a *path-enumeration* search. However, mating search is a very general method for automating proof search; note that it specifies only that one must find a complete mating, without specifying the way in which this is to be achieved. It is therefore possible to define alternative mating search procedures which do not rely on path enumeration; one such procedure is presented in this paper. We first discuss the reasons why one might want to define an alternative search method, and then sketch the details of the new procedure.

Firstly, determining the leftmost open path in a matrix is computationally quite expensive; it would be more efficient to have a search algorithm which could add connections to a mating without searching for open paths, thus separating the construction of the mating from the check for completeness. Furthermore, path-enumeration search becomes increasingly inefficient for large search spaces, since it is essentially a depth-first search of the space of possible matings. The order in which these matings are enumerated depends critically on the order in which vertical paths are enumerated, and hence upon the arrangement of literals in the matrix. In the worst case, if the very first connection which is added is wrong, we must explore all matings which contain this connection before finally backtracking and beginning again essentially from scratch. To compound the problem, one of the best strategies for duplication of quantifiers in a path-enumeration search says (more or less) "don't backtrack until you've tried duplicating all the quantifiers on the current open path"; this makes it even more difficult to backtrack out of blind alleys in the search.

In the higher-order case, unification is also a major problem. Higher-order unification is both undecidable and branching, and in a higher-order mating search the vast majority of the search time is spent on unification rather than mating search. Since our unification algorithm can also introduce negations, unification only gives very weak constraints on the connections which can be added to a mating (essentially, we can unify any connection in which one literal has a higher-order flexible head). This observation, together with the absence of most general unifers, prevents us from introducing many of the control mechanisms and reduction operations which have recently been described in first-order contexts (see, for example, [BBE$^+$95,Brü94]).

The above remarks suggest several problems which our new procedure should address: the problem of devising a breadth-first search strategy (preferably one which is impervious to rearrangements of the matrix), the problem of how to store and reuse as much information as possible (particularly information about higher-order unification), and the problem of avoiding the more expensive parts of the search wherever possible.

In this paper, we describe a search strategy in which small matings containing only a few connections are computed and stored along with their unifiers, and then these units (known as *components*) are combined into larger matings until a complete mating is found. We refer to this strategy as *component* search. We will accept a somewhat inefficient search algorithm in exchange for a much more efficient unification method: instead of incrementally unifying after each connection is added, we unify whole components, store their unifiers as a directed acyclic graph, and then compute the unifiers of larger matings by combining these graphs in an appropriate way. Thus, the main application of component search is to the higher-order case, since in the first-order and propositional cases, this trade-off between unification and search makes little sense: first-order unification can produce a most general unifier, if one exists, in linear time.

Components are similar to the hyper-links of [LP92] (which work only for first-order resolution-based search), while the unifiers of a component can be stored and combined in a manner similar to the ordered binary decision diagrams of [Bry86]. Component search is (almost) breadth-first, avoids path enumeration wherever possible, and can handle many more quantifier duplications than previously possible. Furthermore, the new unification method allows for unification modulo a bounded number of applications of some arbitrary equality rules, a feature not previously available in TPS. In defining component search, we have attempted to separate decisions affecting the search space itself (such as quantifier duplications and set-variable substitutions) from decisions affecting the search for a mating (such as which connection to consider next).

2 Component Search

The following is a well-known result:

Lemma 1 (The Disjunction Heuristic). *If J is a jform containing a disjunction $A \vee B$, and M is a complete and minimal mating for J which uses some literal in A, then M must also use some literal in B.*

Proof. Suppose $c \in M$ is a connection which uses some literal in A. Since the mating is minimal, there must be some vertical path P through J on which c is the only connection (if there were no such path, then $M \setminus \{c\}$ would be a complete mating, contradicting the minimality of M). Let P' be a variant of this path which is identical to P outside of A, but which passes through B instead of A. Since M is complete, P' must also be spanned by some connection $c' \in M$, and clearly $c' \neq c$ since c cannot span P'. If c' does not use any literal from B, then c' spans P, contradicting the assertion that c is the only connection spanning P. Hence c' must use a literal from B. □

This lemma gives us a simple method for extending many matings, and it will form the basis of our search procedure. We now formalise this idea.

Definition 1. *Given a jform J, we define an equivalence relation $\sim_J$ on the literals of J by: $A \sim_J B$ iff A and B lie on all the same horizontal paths (i.e. every horizontal path through* either *A or B passes through* both *A and B).*

It is clear that $\sim_J$ is an equivalence relation, and we will refer to the equivalence classes of $\sim_J$ as the *fragments* of J. For example, if J were in clause normal form, the fragments would correspond exactly to the clauses.

In order to define a search algorithm which is based on fragments, we will require the concepts of *touching* and *blocking*, which we now define.

Definition 2. *Let J be a jform, F a fragment of J, M a mating, and L the set of literals occurring in M. Then*

– *F is* touched by *M iff $F \cap L \neq \emptyset$.*
– *F is* blocked by *M iff $F \subseteq L$.*

We now have the following trivial but essential lemma, which follows immediately from the disjunction heuristic and the definition of $\sim_J$:

Lemma 2. *If M is a minimal complete mating for some jform J, then every fragment touched by M is blocked by M.*

Lemma 2 is the major motivation for the new search procedure, which we shall now outline informally.

Suppose that we are given a jform J, and that there exists some complete mating for J. Suppose further that we have somehow enumerated the acceptable (i.e. unifiable) connections of J, and also that we have enumerated, for each fragment of J, all of the minimal matings which block that fragment.

If we are then given a mating M, we may attempt to extend it to a larger mating by computing the list of fragments $f_1, \ldots, f_n$ which it touches but does not block. By Lemma 2, these fragments must be blocked in any minimal complete mating which extends M. Since we have enumerated all of the minimal matings which block f_1 (suppose they are $C_1, \ldots, C_k$), we may simply generate the new matings $M \cup C_1, \ldots, M \cup C_k$, discarding any new matings which are not unifiable. Each of these new matings is clearly strictly larger than M, and hence if M is in fact a subset of a minimal complete mating, we are guaranteed that one of the new matings is a larger subset of the same minimal complete mating. We may then repeat this process with each of the new matings, until we either reach a complete mating or fail.

Unfortunately, it is possible that there are no fragments which M touches but does not block. This is a necessary (Lemma 2 again) but not sufficient condition for M to be minimal and complete, so at this point we should check for completeness. If the mating is complete, we can stop; if not, we can always find an open path, and we can certainly enumerate all of the connections $c_1, \ldots, c_k$ which block it, and so we can generate $M \cup \{c_1\}, \ldots, M \cup \{c_k\}$ and continue as before. (In fact, we can do a little better than this in some cases, but finding and blocking an open path will remain our last resort.)

If we can somehow get this process started, then we have all the makings of a sound and complete search procedure. To start, we may (for example) pick a fragment using the set of support strategy, and begin with all the minimal sets which block it. If this fragment is touched by any minimal complete mating, we will eventually find a complete mating; if not, then we will eventually fail everywhere, in which case we may choose a new fragment and start again.

Definition 3. *A* component *is a mating M which blocks at least one fragment. A component M is said to be* minimal for the fragment *F iff M blocks F and no proper subset of M blocks F. A* minimal *component is one which is minimal for some fragment.*

Our search algorithm will use the following as a check for completeness of a mating M:

1. If there is a fragment which is touched but not blocked, then the next step of the algorithm should be to block one such fragment.
2. If all the touched fragments are also blocked, it may still be possible to use Lemma 1 to determine some list of fragments, at least one of which must be blocked in any minimal complete mating extending M. (For example, if the jform contains $A \vee (B \wedge C)$, where A, B and C are fragments, and A is blocked, then either B or C must also be blocked.) If we can find such a list, we return it, and at the next step the algorithm should attempt to block all of the given fragments.
3. Otherwise, find an open path and return that. If there is no open path, then M is complete; return `success`.

The first and third steps are essential; the former distinguishes component search from path-enumeration search, and the latter makes sure that component search is complete. We will not discuss the second step further in this paper, for reasons of space. In any case, the second step is non-essential, and its omission will simply mean that one case in the following algorithm never occurs.

The basic search algorithm for a jform J is as follows:

1. Fix all of the search bounds (unification depth if the problem is higher-order, number of quantifier duplications and so on). Amplify J, by applying set-variable substitutions, duplicating outermost quantifiers, and (optionally) hiding any equalities which can be handled by the unification algorithm (see the remarks following Definition 5), to produce a new jform J'.
2. Check each potential connection in J' for unifiability. If it is not unifiable, it will never be considered again; if it is unifiable, then store it as an *acceptable connection.*
3. Divide J' into fragments, and enumerate all of the minimal components for each fragment, discarding any which are not unifiable.
4. Choose one fragment f which has at least one unifiable minimal component. Let *current* be the list of all minimal components for f, and let *stored* be the list of all other components.

5. For each element c of *current*, apply the completeness checker to c:
   (a) If it returns `success`, we have a complete mating; return c.
   (b) If it returns a list of fragments which are touched but not blocked by c, choose one such fragment. Produce a list of new components by taking the union of c with each of the *stored* components which blocks this fragment; discard any elements of this list which are not unifiable.
   (c) If it returns a list of fragments at least one of which should be blocked, then combine c (by taking unions, as above) with each of the stored components which blocks any one of these fragments.
   (d) If it returns an open path, combine c with each of the acceptable connections on this path.
6. If some new components were produced in step 5, set *current* to be the list of components produced, and return to step 5.
7. Otherwise, no minimal complete mating can involve the fragment chosen in step 4. Delete all components touching this fragment, and return to step 4 to choose another starting fragment (unless no components remain, in which case either fail or return to step 1 and expand J' further).

A naïve implementation of this algorithm would be extremely inefficient, and the reader may well wonder why it works at all. The improvements to the unification method will be discussed in Section 3; in the following, we concentrate on the search algorithm itself. Firstly, we attempt to avoid making more than one pass through the outermost loop, by amplifying J many times at step 1; this creates many copies of each quantifier and so we must also introduce methods for controlling the resulting redundancy in the search space, which are discussed in Section 4.

In step 4, we are required to choose a starting fragment; since this fragment will occur in all of the components generated by step 5 (until we either reach step 7 or find a complete mating), we would like to choose a fragment which is very likely to be involved in a minimal complete mating in J'. If we can choose such a fragment, then the loop which begins at step 4 will be completed in one pass. The set of support strategy is one appropriate method for choosing such a fragment. If there are several good candidates, we will choose the one with fewest minimal components, since this will reduce the branching of the search. We also avoid (if possible) choosing any fragment which lies below a set-variable substitution, since this would initially restrict our search to only those matings involving that substitution.

Step 5 of the algorithm above is a strictly breadth-first search; given a mating, it will consider all extensions of that mating during the next step of the procedure. By assigning a weight to each component as it is created (with the intention that a larger weight represents a more promising component, in some sense), and modifying the above algorithm so that only components of maximal weight are extended in step 5, we may force the search procedure to explore the more promising parts of the search space first. This helps to prevent combinatorial explosion of the number of components in the early stages of the search, although the resulting search is of course no longer strictly breadth-first. There are many possible choices for such a weighting, but one which has proven effective in a wide variety of searches is $weight(C)$ = number of fragments blocked by C.

During step 5(b), we must select one fragment which is touched but not blocked. If there is more than one such fragment, then it seems reasonable to select the one which will cause the least branching in the search, which is the one having the fewest minimal components. In practice, we need to take account of the fact that some fragments may be duplicate copies of other fragments, and so we also prefer any original fragment to any duplicate copy, and any nth duplicate copy to any $(n+1)$st duplicate copy.

Finally, we observe that if we are forced to return to step 1 after failing everywhere, it is both possible and desirable to preserve and reuse information from the previous iteration of the search. The information which can usefully be preserved will depend on the exact way in which the search bounds are to be changed in step 1; in particular, if there are changes to the bounds on unification then much less information can be preserved than if the unification bounds remain fixed.

We now turn to the soundness and completeness of this algorithm. For the remainder of this chapter, we assume that unification contains no equality reasoning, and hence that no equalities are hidden by step 1. We also assume, for the sake of clarity in the proof of completeness, that if we reach step 7 and no components remain, we will choose to fail.

Theorem 1 (Soundness). *The above algorithm is sound.*

Proof. If the algorithm terminates without failing (which can only happen at step 5), then the result is a complete and unifiable mating. Hence the procedure is trivially sound, by previous results about the matings method. □

Theorem 2 (Restricted Completeness). *The above algorithm is complete, in the sense that it will find a complete mating in J' if there is one to be found within the given search bounds.*

Proof. (Sketch) At each stage, each component C in *current* is replaced by a set of new components, each of which contains a proper superset of the connections of C. If C was a subset of a minimal mating in J', then at least one of the new components will be a subset of the same minimal mating. Hence it is not possible to repeat step 5 forever; either we will eventually find a complete mating, or we will discard every component that is generated. In the former case, we are done; in the latter case, we will return to step 4. Since, by hypothesis, there is a complete mating M in J', step 4 must eventually choose some fragment f which is blocked by M. □

Note that since there are bounds on the depth of unification, the number of quantifier duplications, and so on, and since we have chosen to fail at step 7 rather than return to step 1, the algorithm is not complete in the more general sense. However, if we suppose that we always opt to return to step 1 (instead of failing), and that all the search bounds are relaxed each time we pass step 1, then we have the following:

Theorem 3 (Completeness). *If there is a complete mating in some expansion of J, then the above algorithm will terminate with a complete mating.*

Proof. (Sketch) Let J^* be the smallest expansion of J in which a complete mating can be found. By soundness, if J' contains no complete mating then the algorithm will eventually return to step 1 and replace J' by some J''. If we have relaxed the search bounds in an appropriate way, then eventually we will reach some jform $J^{(n)}$ which contains a copy of J^* and for which there is a complete mating within the search bounds. Then, by the restricted completeness theorem above, a mating will be found. □

3 Unification

In all of the path-enumeration searches in TPS, unification is performed after each new connection is added. This has the advantage of potentially making unification less expensive (in that the new connection may already be unified, or nearly unified, by the substitutions already generated). However, it also means that the same connection may be unified many times, in many contexts. It also requires unification to backtrack when a connection is discarded. Lastly, very little of the information generated by unification can usefully be stored, since it generally refers only to the immediate situation, which will not recur during the search.

Because of the undecidability of higher-order unification [Hue73,Gol81], we must have bounds on unification during the search. In TPS, these bounds were originally on the depth of the entire unification tree; however, it has been discovered that a uniform bound on the complexity of the substitution for each free variable is a much more efficient approach. It is generally not possible to increase these bounds during the search, since doing so would require reconsidering almost every connection and mating which had previously been discarded on grounds of non-unifiability.

Given this constraint, we can enumerate all of the possible ground substitutions for a given free variable before we even begin the search, using a variant of Huet's algorithm [Hue75,SG89]. This enumeration is possible since we know the maximum depth of any substitution, the types of all the free variables, the constants which occur in the search space, and any new constants that might be required by unification (we will need at most one new constant at each base type, to complete the unification of any remaining flex-flex pairs).

The ground substitutions are used to determine unifiability of a connection in the initial stages of the search. The set of unifiers for a connection or a mating is stored as a directed acyclic graph; when we combine components to produce a larger mating, we can also combine their unification graphs in a natural way to produce a graph containing the unifiers for the larger mating. We now outline the method by which this is achieved.

We assume that a total ordering of the free variables of J is fixed in step 1 of the algorithm. Since we have enumerated the possible substitution terms $t_1, t_2, \ldots, t_n$, we may refer to them by their numbers; we use the number 0 to mean "no substitution". For example, given the variable ordering (x, x', x''), we write $(41, 72, 0)$ for the substitution $\{x \mapsto t_{41}, x' \mapsto t_{72}\}$. There are of course typing restrictions to be satisfied, and the substitutions which occur should be understood to be appropriately typed.

Lemma 3. *Given a variable ordering $(v_1, \ldots, v_k)$, two substitutions $(n_1, \ldots, n_k)$ and $(m_1, \ldots, m_k)$ are compatible iff for all $1 \leq i \leq k$, either $n_i = m_i$ or $n_i m_i = 0$.*

Proof. Since all of the non-zero substitutions are ground, they are pairwise incompatible. Hence two substitutions can only be compatible if they are the same, or one of them is zero. □

Definition 4. *Given a variable ordering $V = (v_1, \ldots, v_n)$ and a list of substitutions S for (some of) the variables in the ordering, the* unification graph $U(V, S)$ *is an arc-labelled directed acyclic graph defined inductively as follows:*

- *If $n = 0$ (V is empty), then $U(V, S)$ is a single node $\perp$ with no descendants.*
- *Otherwise, define the equivalence relation $\sim_S$ on S by $\sigma \sim_S \tau$ iff $\sigma(v_1) = \tau(v_1)$. Let $S_1, \ldots, S_j$ be the equivalence classes of S under $\sim_S$, and for $1 \leq i \leq j$, let m_i be the value of $\sigma(v_1)$ for every σ in the equivalence class S_i. Let $V' = (v_2, \ldots, v_n)$. Now $U(V, S)$ is a node labelled v_1, with j descendant arcs, labelled $m_1, \ldots, m_j$, leading to the nodes $U(V', S_1), \ldots, U(V', S_j)$ respectively.*

We take a fixed variable ordering V of the variables of the jform, and construct a unification graph for each connection C, taking S to be the set of unifying substitutions for C. Each path through a unification graph from top to bottom represents a substitution for V. Our implementation ensures that two unification graphs are equal iff they are the same graph (i.e. `eq` in Lisp), which both saves space and speeds up the search.

For example, suppose $V = (x^1, x^2, x^3)$ and C_1 and C_2 are two connections for which the unifiers are given by $S_{C_1} = \{(1,3,7), (1,4,6), (1,6,8), (2,3,6), (2,4,8)\}$ and $S_{C_2} = \{(1,0,7), (2,3,7), (2,4,0), (3,5,0)\}$ respectively. Then $U(V, S_{C_1})$ and $U(V, S_{C_2})$ are as shown in Figure 1.

We use Lemma 3 to define the *merge* of two unification graphs, which is the intersection of the sets of substitutions they represent. In particular, if C_1 and C_2 are two components having unification graphs U_1 and U_2, then the component $C_1 \cup C_2$ has unification graph *merge*(U_1, U_2). For example, taking C_1 and C_2 as above, the unification graph for the mating $\{C_1, C_2\}$ is the merge of the unification graphs for C_1 and C_2, as in Figure 1.

Unification graphs are very similar to levelized ordered binary decision diagrams, and the algorithm for *merge* is essentially the same as the *apply* algorithm (for the boolean function $\wedge$) given in [Bry86]. The details of the algorithm are omitted for lack of space.

Since unification is done before the mating search begins, we also have a convenient way to unify modulo (some bounded number of applications of) an

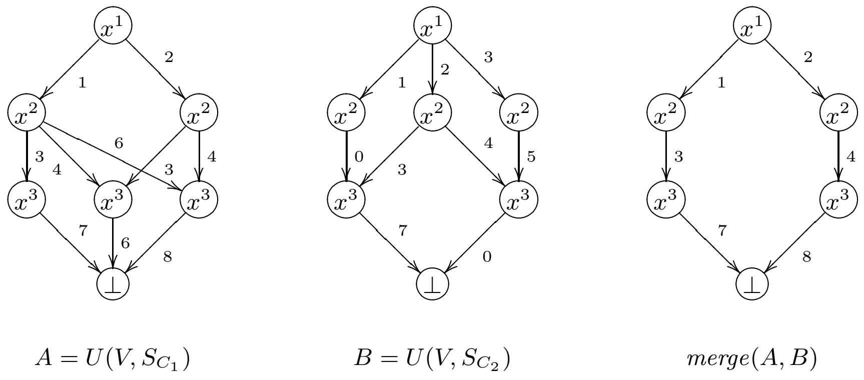


Fig. 1. Some examples of unification graphs.

arbitrary set of equalities. We now outline a method for unification with equalities; the implementation details, and proofs of their correctness, are beyond the scope of this paper.

Definition 5. *Let J be a jform, and $A = B$ an equality*[1] *which occurs positively in J as $\forall q.[\sim qA] \vee qB$. If every vertical path through J passes through either the literal $\sim qA$ or the literal qB, we say that $A = B$ is* global. *If $A = B$ is not global, but every vertical path through a literal L passes through either the literal $\sim qA$ or the literal qB, we say that the equality is* local to L.

When we are performing a search with unification modulo equalities, we will remove all of the global equalities from the jform and restrict the search to whatever literals remain. This can dramatically reduce the size of the search space, the number of quantifier duplications required for a proof, and the minimum unification depth required. We can do this because of the way in which the unification routine is applied: given a pair of literals L_1 and L_2 representing wffs s and t, apply all of the potential unification substitutions σ to the disagreement pair $\langle s, t \rangle$ in the usual way. Now, instead of rejecting any σ for which $\sigma s \neq \sigma t$, take the two ground terms σs and σt and attempt to find a common rewrite for them using (ground instances of) the global equalities and the equalities which are local to both L_1 and L_2. If a common rewrite can be found within the search bounds that have been given, we will accept (L_1, L_2) as a valid connection. In fact, we will implicitly have a chain of connections which begins at L_1 and passes through some number of equalities before eventually reaching L_2; using the definitions of global and local equalities, we can show that this chain of connections will span every path which is spanned by the pair L_1, L_2. When the mating is complete and we attempt to translate it to a natural deduction style proof using the methods of [Pfe87], this chain of connections must be made explicit in order for the translation to work.

[1] Recall that TPS expands equalities using Leibniz' definition: $A = B$ iff $\forall q.qA \supset qB$.

4 Duplication and Set-Variable Instantiation

Previous search procedures in Tps have had two possible approaches to duplication: older procedures waited until all matings had been tried and then duplicated all of the outermost quantifiers, while the newer procedures used path-focused duplication [Iss91] in which the quantifiers on a particular vertical path would be duplicated if there was no other connection available on that path. Path-focused duplication has proven more effective than the older method, but neither approach can handle more than a few quantifier duplications because of the depth-first nature of path-enumeration.

In component search, we perform many duplications (specifically, all the duplications that fall within the search bounds set by the user) before we begin to search for a mating. So, for example, we might duplicate all outermost quantifiers k times before beginning the search. It may appear that, as a strategy for duplication, this is no better than the worst of the former stategies, but in the context of a component search we can control the way in which the new duplications are allowed to appear in a mating, and hence recover the benefits of duplication by need without also recovering its disadvantages.

We distinguish the original copy of a fragment from the first duplication, and that from the second duplication, and so on, and require that a component must touch the first k copies before it is allowed to touch the $(k + 1)$st copy[2]. In this way, we recover the idea of "duplication by need".

We also introduce the concept of equivalence between matings; essentially, two matings M_1 and M_2 are equivalent if there is a permutation of the instances of each quantifier in the jform such that under the induced permutation σ on literals, we have $\sigma(M_1) = M_2$. We then modify the search algorithm to discard new components which are recognisably equivalent to known components (the condition in the previous paragraph is a specific case of this more general restriction). The proof that mating equivalence is well-defined, and that the resulting restrictions on the search algorithm do not affect completeness, are omitted for reasons of space.

We handle set-variable instantiations [BA98,ABI$^+$96] in much the same way as duplications. All of the set-variable instantiations that are to be tried are applied simultaneously at the beginning of the search, and to avoid a combinatorial explosion, the number of such instantiations that can be touched by any particular component is restricted, usually to one or two.

5 Results

Component search was tested on 56 first-order problems which had already been solved by path-enumeration search in Tps. Most of these searches took less than

[2] In fact, a more complicated condition — which we do not have the space to justify in this paper — is necessary: a duplicate fragment created by duplicating some variable x may be touched only if, for every earlier duplicate copy of x, some fragment below that earlier duplicated quantifier has been touched.

a second for both procedures; path-enumeration was generally slightly faster. This was expected, since component search has to spend time enumerating connections and minimal components before it even begins searching.

Results were also compared for 17 previously-solved higher-order problems; component search performed a little better on these problems than did path-enumeration search. However, component search spent a significantly smaller proportion of the total search time on unification than did path-enumeration search (46% of the total time, as opposed to 82% for path-enumeration).

To summarise the results above: on the theorems which could already be proven by TPS, component search is comparable to path-enumeration search. However, the motivation behind component search was to tackle larger search spaces which are intractable for path-enumeration search. The following theorems[3], all proven completely automatically by component search, are intractable for the path-enumeration searches in TPS. All the timings quoted below represent the time taken to find a complete mating on a Dell System Workstation 400MT, with dual 300 MHz Intel Pentium II processors, running TPS under Allegro Common Lisp 4.3 for Linux.

THM126: $\forall h^1_{\beta\gamma} \forall h^2_{\alpha\beta} \forall A_{o\gamma} \forall f^1_{\gamma\gamma\gamma} \forall B_{o\beta} \forall f^2_{\beta\beta\beta} \forall C_{o\alpha} \forall f^3_{\alpha\alpha\alpha}.\text{HOM2 } h^1 A f^1 B f^2 \wedge \text{HOM2 } h^2 B f^2 C f^3 \supset \text{HOM2 } [h^2 \circ h^1] A f^1 C f^3$ *(18.27 seconds)*

HOM2 $hAfBg$ is an abbreviation for "the set A is closed under the binary operation f, likewise B is closed under g, h maps A into B, and for all x and y in A we have $h(f(x,y)) = g(h(x),h(y))$" This theorem states that the composition of two homomorphisms is a homomorphism; an interactive proof of a version of it (formulated in the first-order set theory of von Neumann, Gödel and Bernays) was the central theorem of [BLM+86]. A simpler version, THM133, in which the sets A, B and C are replaced by types α, β and γ (making the closure assumptions unnecessary) is easily provable by TPS, but the search space of THM126 is much larger than that of THM133.

THM145: $\forall R_{o\alpha\alpha} \forall U_{\alpha(o\alpha)}.\text{TRANSITIVE } R \wedge \forall s_{o\alpha}[\forall z_\alpha[sz \supset Rz.Us] \wedge \forall j_\alpha.\forall k_\alpha[sk \supset Rkj] \supset R[Us]j] \supset \forall f_{\alpha\alpha}.\forall x_\alpha \forall y_\alpha[Rxy \supset R[fx].fy] \supset \exists w_\alpha.Rw[fw] \wedge R[fw]w$ *(59.2 minutes)*

This is the Knaster-Tarski fixed point theorem for lattices [Kna27], in a formulation due to Coquand. It states that in a complete lattice, every monotone function f has a fixed point. U is is the least upper bound of the set s, and R is the ordering on the lattice. TPS finds the fixed point $U[\lambda x.Rx[fx]]$.

THM146: $\forall r_{o\iota\iota} \forall x_\iota \forall y_\iota.\forall q_{o\iota\iota}[r \subseteq q \wedge \forall u_\iota \forall v_\iota \forall w_\iota[quv \wedge rvw \supset quw] \supset qxy] \equiv \forall p.r \subseteq p \wedge \text{TRANSITIVE } p \supset pxy$ *(22.5 minutes)*

This theorem states the equivalence of two definitions of the transitive closure of a relation r; it was suggested by Dana Scott. It requires a set-variable instantiation for p of $\lambda u \lambda v.\forall w[ruw \supset qxw] \supset qxv$.

[3] Some remarks about the notation used in TPS: o is the type of truth values, ι is the type of individuals, and $(\alpha\beta)$ is the function type often written $\beta \to \alpha$. A dot corresponds to a left bracket whose mate is as far to the right as is consistent with the brackets already present.

THM270: $\forall f_{\beta\alpha} \forall g_{\xi\alpha} \forall h_{\xi\beta} . \forall x_{\alpha} [h[fx] = gx] \land \forall y_{\beta} \exists x [fx = y] \land$ GEN-HOM $f \land$ GEN-HOM $g \supset$ GEN-HOM h *(0.53 seconds)*
In this theorem, GEN-HOM is an abbreviation for $\lambda f_{\sigma\tau} \forall x_{\tau} \forall y_{\tau} . f[*_{\tau\tau\tau} xy] = *_{\sigma\sigma\sigma}[fx].fy$. Suppose that α, β and ξ are types representing three sets, equipped with binary functions $*_{\alpha\alpha\alpha}, *_{\beta\beta\beta}$ and $*_{\xi\xi\xi}$. If $f_{\beta\alpha}, g_{\xi\alpha}$ and $h_{\xi\beta}$ are functions such that f is surjective, $h \circ f = g$, and both f and g are homomorphisms with respect to the appropriate functions $*$, then h is also a homomorphism. This problem was suggested by Erica Melis.

CR-THEOREM: $\forall R_{o\tau\tau}$.REFLEXIVE $R \land$ TRANSITIVE $R \land$ DIAM R $\supset \forall M_{\tau} \forall N_{\tau}$.EQ* $RMN \supset$ COMMON-REWRITE RMN *(5.08 minutes)*
This is one formulation of the Church-Rosser theorem [CR36]: if R is a reflexive transitive rewrite relation satisfying the diamond property (DIAM: for any A, if A has two rewrites B and C, then B and C have some common rewrite D), and M and N are related by the equivalence closure of R (EQ* R is defined as $\lambda x_{\tau} \lambda y_{\tau} \forall P_{o\tau\tau} . R \subseteq P \land$SYMMETRIC $P \land \forall A_{\tau} \forall B_{\tau} [\exists w_{\tau} [PAw \land PwB] \supset PAB] \supset Pxy$), then M and N have a common rewrite. This formulation is a variant of Theorem 3.1.12 of [Bar84]. The variable P in the definition of EQ* requires a set-variable substitution of the form $\lambda x^1_{\tau} \lambda x^2_{\tau}$COMMON-REWRITE $p_{o\tau\tau} x^1 x^2$, which TPS discovers using the set-variable instantiations of [BA98].

GRP-COMM2: $\forall x_{\iota} [P_{\iota\iota\iota} e_{\iota} x = x] \land \forall y_{\iota} [Pye = y] \land \forall z_{\iota} [Pzz = e] \land \forall x \forall y \forall z [P[Pxy]z = Px.Pyz] \supset \forall a_{\iota} \forall b_{\iota} . Pab = Pba$ *(20.97 seconds)*
If every element of a group is self-inverse, then the group is Abelian.

LATTICE THEORY: Using unification modulo equalities, component search has proven a number of examples in lattice theory (axiomatised in first-order logic with equality), taken from [Szá63]. For TPS, these are good examples of very large search spaces, since the definitions of "lattice", "Boolean algebra", "pentagon" etc. contain many axioms.

1. (Schröder; Thm. 28 of [Szá63]) Join distributes over meet iff meet distributes over join (8.2 min).
2. ([Szá63], p. 122) In a Boolean algebra (complemented distributive lattice), the complement is uniquely defined (4.2 min).
3. (Jordan; Thm. 30 of [Szá63]) The equivalence of two definitions of modularity (6.9 min).
4. (Dedekind; part of Thm. 32 of [Szá63]) A lattice is not modular if it contains a pentagon (6.2 min).
5. (Birkhoff; part of Thm. 33 of [Szá63]) A lattice is not distributive if it contains three points which (taken pairwise) have the same meet and join (5.3 min).

6 Conclusions and Further Work

We have described the component search method, a new mating search algorithm which appears in many cases to be a more powerful search method than path-enumeration search. We have also outlined, as a part of the component search procedure, a graph-based method for storing and manipulating sets of higher-order unifying substitutions. Component search is effective for much larger higher-order search spaces than were previously tractable, since it is less vulnerable to blind alleys in the search space and spends significantly less time on higher-order unification. A more detailed discussion of component search can be found in [Bis99].

Component search seems particularly appropriate for problems in which there are significant parts of the search space which are entirely redundant (for example, proofs from a fixed set of axioms in which some of the axioms are unused in the proof). We make use of this property of the search in applying, before the search begins, many quantifier duplications and set-variable instantiations.

For future work on component search, there are several promising lines of investigation. Improved methods for identifying and pruning redundant matings would allow still larger search spaces to be considered, as would a more intelligent method for choosing a fragment (in the places where the algorithm gives one a choice). There are presumably also better possible weighting functions for components, which would make the search more effective. Even the relation $\sim_J$ which defines the fragments can be changed — under the definition in this paper, there exist jforms in which every fragment contains exactly one literal; a different choice of $\sim_J$ would seem to be called for in these cases.

There are a number of other possible optimizations to the unification procedure, some of which have already been implemented. The definition of variable ordering has been generalised to allow one arc of the unification graph to represent many substitutions, but the influence of the choice of variable order on the search has not yet been studied in any detail. If the analogy with ordered binary decision diagrams is valid, then one would expect the choice of variable order to have a considerable effect on the efficiency of the search.

Lastly, the search algorithm looks like a good candidate for OR-parallelism, in which several processors cooperate on a proof using some kind of blackboard architecture to communicate results about unification graphs and partial matings.

References

ABI$^+$96. Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16:321–353, 1996.

And81. Peter B. Andrews. Theorem proving via general matings. *Journal of the ACM*, 28:193–214, 1981.

And89. Peter B. Andrews. On connections and higher-order logic. *Journal of Automated Reasoning*, 5:257–291, 1989.

BA98. Matthew Bishop and Peter B. Andrews. Selectively instantiating definitions. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 365–380, Lindau, Germany, 1998. Springer-Verlag.

Bar84. H. P. Barendregt. *The λ-Calculus.* Studies in logic and the foundations of mathematics, North-Holland, 1984.

BBE$^+$95. W. Bibel, S. Bruning, U. Egly, D. Korn, and T. Rath. Issues in theorem proving based on the connection method. In Peter Baumgartner, Reiner Hähnle, and Joachim Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods. 4th International Workshop. (TABLEAUX '95)*, volume 918 of *Lecture Notes in Artificial Intelligence*, pages 1–16, Schloß Rheinfels, St. Goar, Germany, May 1995. Springer-Verlag.

Bib82. Wolfgang Bibel. *Automated Theorem Proving.* Vieweg, Braunschweig, 1982.

Bis99. Matthew Bishop. *Mating Search Without Path Enumeration.* PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, 1999.

BLM$^+$86. Robert Boyer, Ewing Lusk, William McCune, Ross Overbeek, Mark Stickel, and Lawrence Wos. Set theory in first-order logic: Clauses for Gödel's axioms. *Journal of Automated Reasoning*, 2:287–327, 1986.

Brü94. S. Brüning. *Techniques for Avoiding Redundancy in Theorem Proving Based on the Connection Method.* PhD thesis, TH Darmstadt, 1994.

Bry86. R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

Chu40. Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

CR36. Alonzo Church and J.B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 3:472–482, 1936.

Gol81. Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

Hue73. Gerard P. Huet. The undecidability of unification in third-order logic. *Information and Control*, 22:257–267, 1973.

Hue75. Gerard P. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

Iss90. Sunil Issar. Path-focused duplication: A search procedure for general matings. In *AAAI–90. Proceedings of the Eighth National Conference on Artificial Intelligence*, volume 1, pages 221–226. AAAI Press/The MIT Press, 1990.

Iss91. Sunil Issar. *Operational Issues in Automated Theorem Proving Using Matings.* PhD thesis, Carnegie Mellon University, 1991. 147 pp.

Kna27. B. Knaster. Une théorème sur les fonctions d'ensembles. *Annales Soc. Polonaise Math.*, 6:133–134, 1927.

LP92. Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyperlinking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.

Pfe87. Frank Pfenning. *Proof Transformations in Higher-Order Logic.* PhD thesis, Carnegie Mellon University, 1987. 156 pp.

SG89. Wayne Snyder and Jean Gallier. Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8:101–140, 1989.

Szá63. Gabor Szász. *Introduction to Lattice Theory.* Academic Press, New York and London, 1963.

The Design of the CADE-16 Inductive Theorem Prover Contest

Dieter Hutter[1] and Alan Bundy[2]

[1] German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
hutter@dfki.uni-sb.de

[2] The University of Edinburgh, Division of Informatics
80 South Bridge, Edinburgh EH1 1HN, Scotland
bundy@dai.ed.ac.uk

"Progress in science is only achieved through careful analysis of methods and their power. Designing without analysis is idle speculation. Implementation without analysis is tinkering. Alone they have no research value. All too often we read of major pieces of AI work that stop with the engineering steps. But we need to know how the implemented program embodies the designs and that the program works well because of the design." B.G. Buchanan [Bu88]

1 Introduction

It has become a tradition at CADE to run a competition for first-order automated theorem provers based on the TPTP problem library. This competition (CASC) [SuSu96] aims at fully automatic ATP systems and provides various categories dedicated to different problem classes of first-order logic. The number of problems solved and the runtime is used to assess the winners in the individual categories of the competition.

In the past considerable effort has been spent to discuss the issues of an inductive theorem proving competition but problems intrinsic to inductive reasoning have prevented a CASC-like competition. Inductive reasoning introduces additional search control problems to those already present in first-order theorem proving. The need for the selection of an appropriate induction order or for the speculation of intermediate lemmata results in infinite branching points in the search space because (technically speaking) the cut rule cannot be eliminated in inductive theories. As a consequence, all current inductive theorem proving systems are necessarily more or less interactive systems. For instance, recursively defined functions can be used to introduce new well-founded orderings to the prover, necessary intermediate lemmata have to be proven as theorems beforehand in order to make them known to the prover. Furthermore, many systems allow the user to complement the problem description by additional strategic information (like the so-called hints in NQTHM [BoMo79]) or allow him to build up a proof in an interactive way. In contrast to first-order theorem proving, inductive theorem proving still depends on the users help. Proving complex inductive theorems depends on the experience and the skills of the human user.

Thus besides their degree of automation, inductive theorem proving systems should also be judged on their abilities to assist the user in the organisation and tracking of complex proof systems [PrSl94].

The time which is necessary to adjust the problem set to the individual inductive theorem provers, to perform the proofs, and to evaluate the results, implies that the competition must take place already before the CADE-conference. Thus, the contest is held over the internet. Various problem sets are emailed to the contest participants. Each participant of the competition treats the problem sets individually and has to provide a detailed report on the behaviour of its system wrt. the individual problems. Each team is also expected to write a detailed report explaining their results and allowing to compare the abilities of the systems wrt. different issues.

The success of an inductive theorem proving system depends on the appropriate formulation of the problems. Different systems provide different aids for the user to control the proof search. They also value differently the individual issues in automating inductive theorem proving. As a consequence, there is no winner of the competition since it seems to be impossible to weigh the different degrees of help, the user has given the system by his individual formulation of the problems. Instead the competition results in a detailed comparison of the participating systems with respect to the several issues. During the FLoC-workshop 13 "Automation of Proofs by Mathematical Induction" selected problems are tackled again (online) by all of the systems in order to study the behaviour of the systems in more detail.

2 Problem Selection and System Evaluation

The competition aims at an evaluation of the capabilities of inductive theorem proving systems with respect to various research problems. A "perfect" system would have to cope with a large variety of issues, such as for instance program synthesis, proving the termination of programs, or providing a sophisticated human-computer interface. Since existing systems usually support only parts of these issues, the selection of specific problem classes to be part of the competition would seriously affect the result of the competition. Hence the design of the competition takes this problem into account by the following means:

The problem set is divided into two categories. The first category aims at "core" issues common to all inductive theorem proving systems. This comprises the selection of an appropriate induction scheme or case analysis and the guidance of the resulting inductive proof obligation. These problems are adapted from a modified version of the NQTHM-92 release. This corpus is based on an $\exists$-quantifier free language and does not incorporate any synthesis problems. Within this category, about a hundred problems from this database are chosen to measure the automation of the various systems. The competition focuses on the number of problems which are solved automatically by the systems and the time they require to solve them.

The second category is dedicated to specific issues of inductive theorem proving mentioned earlier in this paper. Examples are program synthesis (or proving existentially quantified formulas respectively), proving the termination of programs, computing appropriate case analyses and induction schemes, or general issues of first-order theorem proving within inductive proofs. The competition provides specific (but small) problem sets dedicated to these problem classes. In contrast to the first, this category focuses on a comparison *how* different systems tackle these problems. Therefore the participants are requested to fill in a detailed questionnaire to illustrate the degree of automation, the used heuristics and methods, and the limitations of the chosen approach.

3 Problem Presentation

As mentioned before, the competition aims at a variety of different issues in inductive theorem proving. In order to cope with such problems (e.g. computing induction schemes, lemma speculation or proving the termination of algorithms), strategic knowledge of the user is incorporated into the way a particular problem is specified inside a system. Unfortunately the systems vary on the way how (and how much) proof knowledge is given to them. Additionally different provers are based on different logics. While for example NQTHM [BoMo79] is based on a non-sorted logic, CLAM [BvHHS91] uses Martin-Löf type theory. There is no uniform language which suits all of the provers. Thus, the participants are allowed to translate the problem sets into a logic accepted by their system but they are also requested to record this translation in their report.

3.1 Input Language

The principle of induction is strongly correlated to the semantic notion of generated algebras. These generated algebras are typically specified by providing a signature (a set of so-called constructor functions), the terms of which define the objects under consideration. In case of freely generated algebras two different (constructor-) ground terms always denote different objects. For instance natural numbers or lists are typically specified as freely-generated datatypes (based on the signatures $\{0, s\}$ and $\{nil, cons\}$ respectively). The specification of other datatypes like integers or finite sets requires the identification of different constructor-terms. Dealing with non-freely generated datatypes complicates the search for well-founded orderings which is one reason why only a small number of systems allow for these datatypes. Therefore in the first problem category the contest is restricted to freely-generated datatypes. For similar reasons mutually recursive data structures (like for instance terms and termlists) are not used within this category. Nevertheless there are special problems in the second category which are dedicated to these problems.

Inductive theorem proving systems have to create appropriate induction schemes for the given proof obligations. In general this constitutes an infinite branching point. In order to overcome this problem many systems use the recursion ordering of constructively defined functions to formulate appropriate induction schemes. Thus these systems provide schemes to specify functions in a constructive way which provides the systems with both, a set of logical axioms and a specification of well-founded orderings if the denoted algorithm can be proven to be terminating. Although the problem sets makes no use of such specific definition principles but consists of first-order axiomatisation, most of them can be encoded into such a framework. The participants are free to use such definition principles but again have to state this translation in their reports.

4 Conclusion

The CADE-16 contest on inductive theorem proving system is a first approach to compare and evaluate the different systems in this area. The motivations for running such a competition are to evaluate inductive theorem provers, to identify possible improvements and open research problems, and to construct a problem library for inductive theorem provers. Although we are aware of the limitations of this contest we see a clear potential for future competitions which will be improved with respect to the problem library and also with respect to a more granular evaluation of the experimental results.

Acknowledgements. We would like to thank David McAllester for initiating the discussion about a contest and for providing a first version of the problem library. We are also indebted to Ian Green who translated this first version of a library into a sorted language.

References

BoMo79. B. Boyer and J S. Moore A Computational Logic. *Academic Press*, 1979

Bu88. B.G. Buchanan: Artificial Intelligence as an Experimental Science. *In J.H. Fetzer (ed.): Aspects of Artificial Intelligence*, pp. 209-250, Kluwer Academic Publishers, 1988

BvHHS91. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303-324, 1991

PrSl94. C. Prehofer and K. Slind: Theorem Proving in Interactive Verification Systems. *Proceedings of the CADE-12 Workshop: Evaluation of Automated Theorem Proving Systems*, 1994

SuSu96. C. Suttner and G. Suttcliff: The Design of the CADE-13 ATP System Competition. *Proceedings of the 13th International Conference on Automated Deduction, CADE-13*, Springer LNAI 1104, New Brunswick, USA, 1996

System Description: SPASS Version 1.0.0

Christoph Weidenbach, Bijan Afshordel, Uwe Brahm, Christian Cohrs,
Thorsten Engel, Enno Keen, Christian Theobalt, and Dalibor Topić

Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany
`weidenb@mpi-sb.mpg.de`

Abstract. SPASS is an automated theorem prover for full first-order logic with equality. This system description provides a rough overview over recent developments.

1 Introduction

SPASS is an automated theorem prover for full sorted first-order logic with equality that extends superposition by sorts and a splitting rule for case analysis (Weidenbach, Gaede & Rock 1996). The prover is implemented in C. We provide binaries for many platforms and the prover should be portable to any platform where a C-compiler is available.

SPASS is meant to be useful for two groups of users. People using it as a tool and people using it as a development platform. Many of our recent efforts went into improving ease of use for both groups.

2 New Features

Compared to the version of SPASS described earlier by Weidenbach et al. (1996) nearly all existing code was re-implemented or adjusted and among others we added the features described below.

2.1 Code Development

One of our long term goals is to provide as much as possible of the functionality of SPASS in form of a documented C-library. Towards this end, we improved modularization, debugging support and code documentation. Except from the documentation contained in the code, there is currently not much extra documentation available, except for our memory management module. The module allocates memory in pages of uniform size that are then cut into appropriate pieces. In addition to a gain in performance, the module supports many debugging features that are otherwise only supported by commercial software. For example, it can detect writes over memory blocks or can point to memory leaks.

For SPASS we have now adopted the GNU command line options package for C. So all SPASS options can now be given as command line options to SPASS. In addition to options selecting inference or reduction rules, the selection of various strategies like, e.g., set of support and various possibilities to influence the way SPASS's output looks like, there is the possibility to feed SPASS via pipes and to use the prover in a kind of interactive way: First, a set of axioms is given to SPASS and then the prover can be subsequently given conjectures to prove with respect to such an axiom set. This is particularly useful in a context where SPASS is integrated as an inference engine for some other system.

We added an abstract ordering interface module to SPASS, where in particular the lifting of an arbitrary reduction ordering to literals/clauses is implemented. In addition to the Knuth-Bendix ordering (KBO) implemented in Version 0.42, we added an implementation of the recursive path ordering with status (RPOS).

The input language of SPASS was changed to DFG-Syntax that can represent formulas, clauses and proofs (Hähnle, Kerber & Weidenbach 1996). A description of the syntax together with some example/benchmark problem libraries is available from the SPASS homepage (see 2.9).

2.2 Clause Normal Form Translation

SPASS now has the clause normal form translation built in, so it can be directly applied to first-order formulae and it can be used to convert formulae into CNF. The current implementation contains all features described by Nonnengart, Rock & Weidenbach (1998): Optimized and strong Skolemization and the improved implementation of formula renaming, the replacement of subformulae by new predicate symbols. Furthermore, we extended renaming such that it now first searches for generalizations of renaming candidates and then simultaneously replaces all instances of the found generalization. We say that a formula ϕ is an instance of a formula ψ if there exists a substitution σ for the free variables in ψ such that $\psi\sigma$ is a renamed variant of ϕ.

For problems containing equality we added a bunch of simplification rules that eliminate occurrences of equations. For example, the rule

$$\forall x\, [x \approx t \supset \psi] \rightarrow \psi\{x \mapsto t\}$$

can be used to remove the equation $x \approx t$ if x does not occur in t. For some classes of problems, cardinality properties of minimal models are known or can be easily derived. For example, if we know that any minimal model of some formula has at least two domain elements, the rule

$$\forall x\, (x \not\approx t \wedge \phi) \rightarrow \bot$$

falsifies a complete (sub)formula if x does not occur in t.

2.3 Atom Definitions

It is often useful to expand atom definitions before CNF transformations and/or to apply atom definitions to conjecture formulae/clauses. An atom definition is meant to be formula of the form

$$\forall x_1, \ldots, x_n[\phi \supset (P([x_1, \ldots, x_n]) \equiv \psi)]$$

where $P([x_1, \ldots, x_n])$ denotes an arbitrary atom with predicate symbol P containing the variables $x_1, \ldots, x_n$. We require that P does not occur in ψ. SPASS searches an input file for formulae that can be transformed in the above form and then allows the user via options to replace occurrences of atoms $P([t_1, \ldots, t_n])$ by $\psi\sigma$ if in the replacement context the formula $\phi\sigma$ is valid where $\sigma = \{x_i \mapsto t_i \mid 1 \leq i \leq n\}$.

2.4 Inference Rules

We added a bunch of new inference rules to SPASS: Ordered/unordered hyper resolution, unit resolution, merging paramodulation, ordered/unordered paramodulation. All inference rules can be combined with selection strategies for negative literals. A further new inference rule is depth bounded unit resolution, a variant of unit resolution that requires the term depth of a unit resolvent to be less or equal to the maximal term depth of its parents. Although this rule is not complete, even in a Horn setting, it is guaranteed to terminate on any clause set and turned out to be very useful in practice for subproofs in the context of optimized Skolemization (see 2.2, (Nonnengart et al. 1998)) and the applicability test for definitions (see 2.3).

2.5 Transformation/Approximation

For many applications, like, e.g., automatic type inference it is necessary/useful that the prover is a decision procedure for the input formula classes. If classes do not belong to a decidable fragment, safe approximations can be used to guarantee termination. We concentrated on monadic clause classes and implemented various methods to transform arbitrary clause sets into monadic clause sets. These can then be further approximated into decidable monadic clause sets. For example, a clause

$$\neg R(x, f(x,y)) \vee \neg S(x) \vee R(g(x), h(y))$$

can first be equivalently transformed into a monadic clause

$$\neg T(r(x, f(x,y))) \vee \neg S(x) \vee T(r(g(x), h(y)))$$

and then be approximated by the clauses

$$\neg T(r(x, f(x,y))) \vee \neg S(x) \vee \neg P(z) \vee \neg Q(v) \vee T(r(z,v))$$
$$\neg T(r(x, f(x,y))) \vee \neg S(x) \vee P(g(x))$$
$$\neg T(r(x, f(x,y))) \vee \neg S(x) \vee Q(h(y))$$

that then overestimate the relation R, represented by the function r in the approximation. The fragment formed by clauses of the final form is decidable and effective representations for R can be derived by a saturation of the clause set (Weidenbach 1999). Approximation techniques are also a prerequisite for semantic approaches to guide the search for a prover itself (Ganzinger, Meyer & Weidenbach 1997).

2.6 Proof Documentation/Checking

We completely re-implemented the extraction of proofs out of SPASS runs. The extracted proofs are now less redundant with respect to the application of the splitting rule and our proof module is now able to deal with proofs of several hundred thousand steps in reasonable time. If splitting occurs in a proof, the proof has a tree (tableau) like structure. It is now possible to transform such proofs into trees that can then be graphically displayed.

Furthermore, we built an automated proof checker based on logical implication between clauses. SPASS proofs are translated into a sequence of proof obligations, one for each proof step. The tree structure caused by splitting applications is separately checked by an independent algorithm. Every single proof step obligation is then checked by a (different) prover. We usually employ Otter for this task. The advantage of this method is that it is completely independent from variations of the used calculus (see 2.4) and is able to check proofs up to several hundred thousand steps in reasonable time. Since the proof checker does not depend on SPASS nor on the calculus used in SPASS, any clause based refutation proof that relies on logical implication can be checked using the checker.

2.7 WWW Interface

We added a www interface to the SPASS homepage (see 2.9). The interface includes the possibility for a file upload and offers the full functionality of the prover including help files. In order to restrict the load of the server, SPASS runs are currently limited to 30 seconds.

2.8 Tools

We added a bunch of tools:

FLOTTER	is still our CNF-translator. Now its simply implemented by a link to SPASS.
`pcheck`	is a proof checker (see 2.6).
`dfg2otter`	transforms SPASS input files into Otter syntax. Our motivation for this was to employ Otter as a proof checker (see 2.6).
`dfg2tptp`	transforms SPASS input files into TPTP-Syntax.
`prolog2dfg`	transforms prolog programs into SPASS input files. This is currently restricted to purely logical programs. The motivation is type inference from PROLOG programs using SPASS (see also 2.5).
`dfg2ascii`	this tool provides an ASCII pretty print variant for SPASS input files.
`dfg2dfg`	this is a conversion tool that transforms SPASS input files, currently dedicated to the computation/approximation of monadic clause classes (see 2.5).

A combination of the translator `dfg2tptp` with `tptp2X` yields a translation procedure from DFG-Syntax into all prover formats supported by the TPTP-library (Sutcliffe & Suttner 1998).

2.9 Distribution

The Spass distribution now also contains binaries for SUN Sparc Solaris, DEC Alpha Ultrix, PC Linux, PC X86 Solaris and PC Windows 95/Windows 98/Windows NT. For the Windows versions we added a neat GUI to SPASS that is built using the Qt library and also available as a SUN Sparc Solaris binary. Under a unix environment, the source code (excluding the GUI) should compile without any problems if recent versions of the standard GNU tools `bison`, `flex`, `make` and `gcc` are available. The distribution is available from the Spass homepage:

`http://spass.mpi-sb.mpg.de/`

where also links to the WWW-Interface, documentation and problem libraries exist.

The distribution of Spass contains texinfo based documentation in different formats: man pages, info pages, html pages and postscript documents.

3 Future Directions

Further development of Spass will concentrate on application domains like security protocol analysis, software verification and automatic type inference.

References

Ganzinger, H., Meyer, C. & Weidenbach, C. (1997), Soft typing for ordered resolution, *in* ‘Proceedings of the 14th International Conference on Automated Deduction, CADE-14’, Vol. 1249 of *LNAI*, Springer, Townsville, Australia, pp. 321–335.

Hähnle, R., Kerber, M. & Weidenbach, C. (1996), Common syntax of the DFG-Schwerpunktprogramm “Deduktion”, Interner Bericht 10/96, Universität Karlsruhe, Fakultät für Informatik, Germany.

Nonnengart, A., Rock, G. & 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.

Sutcliffe, G. & Suttner, C. B. (1998), ‘The tptp problem library – cnf release v1.2.1’, *Journal of Automated Reasoning* **21**(2), 177–203.

Weidenbach, C. (1999), Towards an automatic analysis of security protocols in first-order logic, *in* H. Ganzinger, ed., ‘16th International Conference on Automated Deduction, CADE-16’, LNAI, Springer. This volume.

Weidenbach, C., Gaede, B. & Rock, G. (1996), Spass & flotter, version 0.42, *in* M. McRobbie & J. Slaney, eds, ‘13th International Conference on Automated Deduction, CADE-13’, Vol. 1104 of *LNAI*, Springer, pp. 141–145.

Kꓘ: A Theorem Prover for K

Andrei Voronkov

Computer Science Department, University of Manchester

1 General Description

Nonclassical propositional logics play an increasing role in computer science. They are used in model checking, verification, and knowledge representation. Traditional decision procedures for these logics are based on semantic tableaux [6,7,1], SAT-based methods [4], or translation into classical logic [10]. In this system abstract we overview the system Kꓘ that implements the tableau method and the less traditional inverse method for propositional modal logic K.

The main purpose of Kꓘ is to compare top-down (tableau) and bottom-up (inverse) decision methods for K and other logics. It is expected to extend Kꓘ with several modal logics, description logics and quantified boolean formulas.

The theoretical basis for Kꓘ is presented in [13]. The system is implemented in ML and runs under Windows32 and Unix. No graphical interface is yet available.

Propositional K is PSPACE-complete. Tableau-based decision procedures for K can easily be implemented in polynomial space. It is not clear whether the inverse method can, in general, be implemented in polynomial space[1].

2 Preliminaries

For simplicity, we restrict ourselves to formulas built up from literals (i.e. propositional variables or their negations) using $\wedge$, $\vee$, $\Box$ and $\Diamond$. We denote propositional variables by lower-case letters, and formulas by upper-case letters. A *sequent* is a finite multiset of formulas. We denote sequents by Greek letters Γ and Δ. For a sequent $\Gamma = A_1, \ldots, A_n$, we denote by $\Diamond\Gamma$ the sequent $\Diamond A_1, \ldots, \Diamond A_n$. A sequent $A_1, \ldots, A_n$ is called *satisfiable* if it is true in some Kripke model and *unsatisfiable* otherwise.

3 The Tableau Method Implementation

The tableau method implemented in Kꓘ is based on the sequent calculus Seq given in Figure 1. The sequent calculus is complete for unsatisfiability: a formula A is unsatisfiable if and only if it has a proof in Seq.

[1] Recently, Hustadt and Schmidt [10] have shown that semantic tableaux can be polynomially simulated by a variant of resolution, but we do not have enough details to comment on the relevance of their results for the inverse method.

H. Ganzinger (Ed.): CADE-16, LNAI 1632, pp. 383–387, 1999.

$$\text{axioms: } \Gamma, p, \neg p \qquad \frac{\Gamma, A, B}{\Gamma, A \wedge B}\ (\wedge)$$

$$\frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \vee B}\ (\vee) \qquad \frac{\Delta, A}{\Gamma, \Box\Delta, \Diamond A}\ (\Diamond)$$

Fig. 1. Sequent calculus Seq

The proof-search by the tableau method is implemented in a straightforward way as a backward proof procedure for Seq. We use an efficient encoding of sequents and the heuristics selecting the shortest subformula in a sequent. We also tried to add to Seq a restricted but sound version of simplification rules

$$\frac{\Gamma[p \leftarrow \text{true}]}{\Gamma, p} \qquad \frac{\Gamma[p \leftarrow \text{false}]}{\Gamma, \neg p}$$

but it did not improve the performance.

4 The Inverse Method Implementation

The inverse method is based on a bottom-up version of Seq. The sequent calculus for the inverse method depends on the goal formula G, is denoted by Inv_G and shown in Figure 2.

$$\text{axioms: } p, \neg p \qquad \frac{\Gamma, A, A}{\Gamma, A}\ (C)$$

$$\frac{\Gamma, A}{\Gamma, A \wedge B}\ (\wedge_l) \qquad \frac{\Gamma, B}{\Gamma, A \wedge B}\ (\wedge_r)$$

$$\frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \vee B}\ (\vee)$$

$$\frac{\Gamma, A}{\Box\Gamma, \Diamond A}\ (\Diamond) \qquad \frac{\Gamma}{\Box\Gamma, \Diamond A}\ (\Diamond')$$

All formulas occurring in the rules are subformulas of G.

Fig. 2. Sequent calculus Inv_G

This calculus is complete in the following sense: a formula G is unsatisfiable if and only if it is derivable in Inv_G (completeness for sequents can also be proved but is formulated in a different way). The implementation of the inverse method is based on a standard saturation procedure:

1. Let S be a set of sequents, initially $\{p, \neg p \mid p, \neg p$ 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 derivation of G is found.

However, such a procedure is hopelessly inefficient compared to the tableau method, unless augmented with powerful redundancy criteria allowing us to get rid of redundant sequents and redundant inferences.

Several such redundancies are discussed in [13]. The redundancies used in our implementation are obtained by analyzing possible derivations in Seq and specializing Inv_G to avoid derivations that either cannot occur in Seq or correspond to redundant derivations in Seq. We also used subsumption as a redundancy criterion.

5 Experiments

In this section we consider experimental results obtained by running our system on 20 randomly generated benchmarks. The benchmarks were generated using the random formula generator implemented by Fabio Massacci[2] and discussed in [8].

Since our aim was a prototype implementation and the comparison of the inverse method with the tableau method, our current implementation is made in ML. A more efficient implementation in C++ will be released soon.

The results (in seconds, rounded down to an integer) are given in Table 1. The second column shows the results for the tableau method, the third column for the inverse method. The fourth column describes the naive combination of the two methods: two methods are run in parallel using equal system resources until at least one of them terminates. The bottom rows of the table give the average time for all 20 problems, the average time for 80% of the problems (excluding two fastest and two slowest) and the number of problems solved within some time intervals (for example, 1 minute). The best results are grayed.

We would like to emphasize that we compare a *very straightforward* implementation of the inverse method with a very straightforward implementation of the tableau method. Some remarks about further possible optimizations are included in Section 6.

As in the case of CASC competition for classical logic, one can note that the correlation between runtimes for different methods is quite loose. We do not want to discuss the big difference in performance on some benchmarks (though there is an easy explanation), for example problem 11 is solved in less than a second by the tableau method and in about 7 hours by the inverse method. Our implementations of both methods are not optimized which causes instability of experimental results.

[2] `http://hermes.dis.uniroma1.it/~massacci/TANCS/problem.html`. The generator was called with the -K option.

problem	tableau	inverse	mixed
1	1	4	2
2	403	1	2
3	1,296	913	1,826
4	12,642	1,264	2,528
5	10,323	1,135	2,270
6	4,383	4	8
7	2,120	65	130
8	16,332	8,477	16,954
9	102,339	4,187	8,374
10	7	23	14
11	0	23,198	0
12	3,690	17,512	7,380
13	15	586	30
14	910	21	42
15	275	11	22
16	31	789	62
17	23,290	1,370	2,740
18	11,626	46	92
19	701	20	40
20	1,348	72	144
avg	9,586	2,984	2,576
80%	4,131	1,186	1,083
10 sec	3	3	4
1 min	5	8	9
5 min	6	10	13
30 min	11	16	13

Table 1. Experimental results

6 Further Developments

To implement the inverse method more efficiently, more investment in datastructures and algorithms in needed. ML is not very efficient (both in terms of time and space) for such datastructures. We are now implementing a C++ version of the prover. Many important optimizations have not been included in the system yet.

When we implemented the first version of Kꓘ, we were not aware of the optimizations used in the fastest tableau-based implementations and described in [7]. All redundancy criteria used to optimize our implementation are specific for the inverse method and cannot be used for tableaux. We are currently investigating which optimizations described in [7] can be applied to the inverse method. Our guess is that any optimizations not destroying the subformula property can be used in some form without losing completeness. Some optimizations of [7] do not preserve the subformula property, but in a very liberal way. For example, in the semantic splitting rule

$$\frac{\Gamma, A \quad \Gamma, \neg A, B}{\Gamma, A \vee B}\ (\vee)$$

the formula $\neg A$ may not be a subformula of the goal formula. However, it is not difficult to modify the inverse method by allowing negations of all subformulas to be used.

Acknowledgments

Part of this research was done in Uppsala University, where the author was supported by TFR grants. We thank Larisa Maksimova for helpful comments on modal logic.

References

1. F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithms. In H.Boley and M.M.Richter, editors, *Processing Declarative Knowledge (Proceedings of the International Workshop PDK'91*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 67–86. Springer Verlag, 1991.
2. L. Catach. Tableaux: a general theorem prover for modal logics. *Journal of Automated Reasoning*, 7(4):489–510, 1991.
3. M. D'Agostino. Are tableaux an improvement of truth tables? *Journal of Logic, Language and Information*, 1:235–252, 1992.
4. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures: Case study of modal K. In M.A. McRobbie and J.K. Slaney, editors, *CADE-13*, volume 1104 of *Lecture Notes in Computer Science*, pages 583–597, 1996.
5. L.F. Goble. Gentzen systems for modal logic. *Notre Dame J. of Formal Logic*, 15:455–461, 1974.
6. I. Horrocks and P.F. Patel-Schneider. FaCT and DLP. In H. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX'98*, volume 1397 of *Lecture Notes in Computer Science*, pages 27–30, Oisterwijk, The Netherlands, May 1998. Springer Verlag.
7. I. Horrocks and P.F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 1999. To appear.
8. U. Hustadt and R. Schmidt. On evaluating decision procedures for modal logic. In *IJCAI-97*, volume 1, pages 202–207, 1997.
9. U. Hustadt and R.A. Schmidt. Simplification and backjumping in modal tableau. In H. de Swart, editor, *TABLEAUX'98*, volume 1397 of *Lecture Notes in Computer Science*, pages 187–201, 1998.
10. U. Hustadt and R.A. Schmidt. On the relation of resolution and tableaux proof systems for description logics. In *IJCAI-99*, 1999. To appear.
11. S.Yu. Maslov. An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus. In J.Siekmann and G.Wrightson, editors, *Automation of Reasoning (Classical papers on Computational Logic)*, volume 2, pages 48–54. Springer Verlag, 1983.
12. A. Voronkov. Theorem proving in non-standard logics based on the inverse method. In D. Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 648–662, 1992.
13. A. Voronkov. A bottom-up decision procedure for propositional K: theory and implementation. 1999. To be submitted.

System Description: *CYNTHIA*

Jon Whittle[1][⋆], Alan Bundy[1], Richard Boulton[1], and Helen Lowe[2][⋆⋆]

[1] Division of Informatics, University of Edinburgh, 80 South Bridge
Edinburgh EH1 1HN, Scotland
[2] Dept. of Computer Studies, Glasgow Caledonian University
City Campus, Cowcaddens Road, Glasgow G4 0BA, Scotland
jonathw@ptolemy.arc.nasa.gov

Current programming environments for novice functional programming (FP) are inadequate. This paper describes ways of using proofs as a foundation to improve the situation, in the context of the language ML [4]. The most common way to write ML programs is via a text editor and compiler (such as the Standard ML of New Jersey compiler). But program errors, in particular type errors, are generally difficult to track down. For novices, the lack of debugging support forms a barrier to learning FP concepts [5].

CYNTHIA is an editor for a subset of ML that provides improved support for novices. Programs are created incrementally using a collection of correctness-preserving editing commands. Users start with an existing program which is adapted by using the commands. This means fewer errors are made. *CYNTHIA*'s improved error-feedback facilities enable errors to be corrected more quickly. Specifically, *CYNTHIA* provides the following correctness guarantees:

1. syntactic correctness;
2. static semantic correctness, including type correctness as well as checking for undeclared variables or functions, or duplicate variables in patterns etc.;
3. well-definedness — all patterns are exhaustive and have no redundant matches;
4. termination.

Note that, in contrast to the usual approach, correctness-checking is done incrementally. Violations of (1), (3) and (4) can never be introduced into *CYNTHIA* programs. (2) may be violated as in general it is impossible to transform one program into another without passing through states containing such errors. However, all static semantic errors are highlighted to the user by colouring expressions in the program text. The incremental nature of *CYNTHIA* means that as soon as an error is introduced, it is indicated to the user, although the user need not change it immediately.

In *CYNTHIA*, each ML function definition is represented as a proof of a specification of that function, using the idea of proofs-as-programs [2]. As editing commands are applied, the proof is developed hand-in-hand with the program, as given in Fig. 1. The user starts with an existing program and a corresponding

[⋆] This author is now with Recom Technologies, NASA Ames Research Center, Moffet Field. CA 94035

[⋆⋆] First author supported by EPSRC studentship and 519-50-32 NASA Code R Aero IT Base Program, SIPS, Program Synthesis. Other authors EPSRC grant GL/L/11724 initial proof (from an initial library). The edits are actually applied to the proof, giving a new partial proof which may contain gaps or inconsistencies. C^YNTHIA attempts to fill these gaps and resolve inconsistencies. Any which cannot be resolved are fed back to the user as program errors.

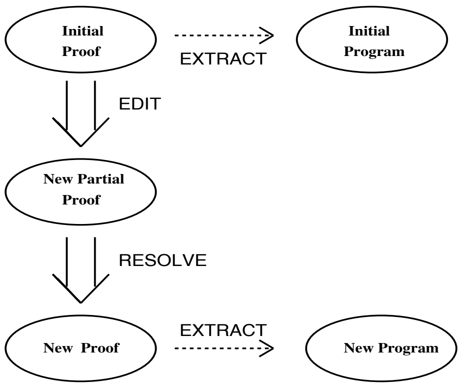


Fig. 1. Editing Programs in C^YNTHIA.

C^YNTHIA's proofs are written in *Oyster* [1], a proof-checker implementing a variant of Martin-Löf Type Theory. *Oyster* specifications (or conjectures) may be written to any level of detail, but to make the proof process tractable in real-time, C^YNTHIA specifications are restricted severely. Specifications state precisely the type of the function and various lemmas needed for termination analysis. Proofs of such specifications provide guarantees (1)-(4) above. Given this restriction, all theorem proving can be done automatically.

We only consider a functional subset of the Core ML language [5]. In addition, we exclude mutual recursion and type inference. Mutual recursion could be added by extending the termination checker. We made a conscious decision to insist that the user provide type declarations. This is because the system is primarily intended for novices and investigations have shown that students find type inference confusing [5].

1 An Example of Using C^YNTHIA

To illustrate the idea, consider the task of writing a function, `count`, to count the number of nodes in a binary tree, where the definition of the datatype `tree` is given in ML as:

```
datatype tree = leaf of int | node of int * tree * tree;
```

Suppose the user recognises that a function, `length`, to count the number of items in an integer list, is similar to the desired function. He[1] can then use `length` as a starting point. Below we give the definition of `length` preceded by its type

[1] We refer to the user by the pronoun 'he' although the user may be male or female.

```
'a list -> int
fun length nil = 0
|   length (x::xs) = 1 + (length xs);
```

Note that `'a list` is the polymorphic list type. We show how `length` could be edited into `count`. The user may indicate any occurrence of `length` and invoke the RENAME command to change `length` to `count`. *CYNTHIA* then changes all other occurrences of `length` to `count`:

```
'a list -> int
 fun count nil = 0
 |   count (x::xs) = 1 + (count xs);
```

We want to count nodes in a tree so we need to change the type of the parameter. Suppose the user selects `nil` and invokes CHANGE TYPE to change the type to `tree`. *CYNTHIA* propagates this change by changing `nil` to `leaf n` and changing `::` to `node`:

```
tree -> int
fun count (leaf n) = 0
|   count (node(x,xs,ys)) = 1 + (count xs);
```

A new variable `ys` of type `tree` has been introduced. In addition, `count ys` is made available for use as a recursive call in the program. It remains to alter the results for each pattern. `0` is easily changed to `1` using CHANGE TERM. If the user then clicks on `1` in the second line, a list of terms appear which include `count ys`. Selecting this term produces the final program:

```
tree -> int
fun count (leaf n) = 1
|   count (node(x,xs,ys)) = 1 + (count ys) + (count xs);
```

2 Representing ML Definitions as Proofs

Each ML function is represented by a proof with specification (i.e. top-level goal) that is precisely the type of the function along with lemmas required for termination analysis. In general, such specifications may specify arbitrarily complex behaviour about the function. However, *CYNTHIA* specifications are deliberately rather weak so that the theorem proving task can be automated. The definition of quicksort given in Fig. 2 is easily represented in *CYNTHIA*.

A specification for `partition` would be as follows:

$$\exists P : (\forall z_1 : (int * int \rightarrow bool). \forall z_2 : int. \forall z_3 : int\ list.$$
$$(f\ z_1\ z_2\ z_3) : int\ list \wedge (f\ z_1\ z_2\ z_3) \leq_w z_3) \qquad (1)$$

where f represents the name of the function and P is a variable representing the definition of the ML function. P gets instantiated as the inference rules are applied. A complete proof instantiates P to a complete program. This is a standard approach to extracting programs from proofs. The first part of the

```
(int * int -> bool) -> int -> int list -> int list
fun partition f k nil = nil
|  partition f k (h::t) = if f(h,k) then h::partition f k t
                          else partition f k t;



int list -> int list
fun qsort nil = nil
|  qsort (h::t) = (qsort (partition (op <) h t)) @ [h]
                       @ (qsort (partition (op >=) h t));
```

Fig. 2. A Version of Quicksort.

(weak) specification merely states the existence of a function of the given type. The last conjunct specifies a termination condition.

$C^Y NTHIA$'s termination analysis is an extension of Walther Recursion [3] to ML. Walther Recursive functions form a decidable subset of the set of terminating functions, including primitive recursive functions over an inductively-defined datatype, nested recursive functions and some functions with previously defined functions in a recursive call, such as `qsort`. Walther Recursion assumes a fixed size ordering: $w(c(u_1, \ldots, u_n)) = 1 + \sum_{i \in R_c} w(u_i)$ where c is a constructor and R_c is the set of recursive arguments of c.

There are two parts to Walther Recursion — reducer / conserver (RC) analysis and measure argument (MA) analysis. Every time a new definition is made, RC lemmas are calculated for the definition. These place a bound on the definition based on the fixed size ordering. In (1), a conserver lemma for `partition` is $(f\ z_1\ z_2\ z_3) \leq_w z_3$ which says that the result of `partition` has size no greater than its third argument. To guarantee termination, it is necessary to consider each recursive call of a definition and show that the recursive arguments decrease with respect to this ordering. Since recursive arguments may in general involve references to other functions, showing a measure decrease may refer to previously derived RC lemmas. In `qsort`, for example, we need, amongst other things, to show that `partition (op >=) h t` $\leq_w$ `t`. This is achieved by using the lemma for `partition`.

Walther Recursion is particularly appropriate because $C^Y NTHIA$ is meant for novices who have no knowledge of theorem proving. The set of Walther Recursive functions is decidable. Hence, termination analysis is completely automated.

Clearly, there are an infinite number of proofs of a specification such as (1). The particular function represented in the proof is given by the user, however, since each editing command application corresponds to the application of a corresponding inference rule. In addition, many possible proofs are outlawed because the proof rules (and corresponding editing commands) have been designed in such a way as to restrict to certain kinds of proofs, namely those that correspond to ML definitions.

Formula (1) can be proved in a backwards fashion. The main ingredients of this proof are induction, to represent the recursion in `partition`, along with various correctness-checking rules which perform type-checking, termination analysis etc. The structure of the program is mirrored in the proof because the user drives the proof indirectly by applying editing commands to the program.

The use of proofs to represent ML programs is a flexible framework within which to carry out various kinds of analyses of the programs. It allows changes at multiple places in the program to be achieved by a single change to the proof, e.g. the induction scheme captures the recursion pattern of the function.

3 Evaluating *C^Y NTHIA*

C^Y NTHIA has been successfully evaluated in two trials at Napier University [5]. Although some semi-formal experiments were undertaken, most analysis was done informally. However, the following trends were noted:

- Students make fewer errors when using *C^Y NTHIA* than when using a traditional text editor.
- When errors are made, users of *C^Y NTHIA* locate and correct the errors more quickly. This especially applies to type errors.
- *C^Y NTHIA* discourages aimless hacking. The restrictions imposed by the editing commands mean that students are less likely, after compilation errors, to blindly change parts of their code.

C^Y NTHIA can be downloaded from http://ase.arc.nasa.gov/whittle/

References

1. A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.
2. W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry; Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.
3. David McAllester and Kostas Arkoudas. Walther recursion. In M. A. McRobbie and J. K. Slaney, editors, *13th International Conference on Automated Deduction (CADE13)*, pages 643–657. Springer Verlag LNAI 1104, July 1996.
4. R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML.* MIT Press, 1990.
5. J.N.D. Whittle. *The Use of Proofs-as-Programs to Build an Analogy-Based Functional Program Editor.* PhD thesis, Division of Informatics, University of Edinburgh, 1999.

System Description: MCS: Model-Based Conjecture Searching*

Jian Zhang

Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing 100080, CHINA
zj@ox.ios.ac.cn

Abstract. An enumerative approach to conjecture formulation is presented. Its basic steps include the generation of terms and formulas, and testing the formulas in a set of models. The main procedure and some experimental results are described.

1 Introduction

Automated theorem provers are used to prove conjectured theorems. But where do the conjectures (theorems) come from? In most cases, they are provided by human users. However, a user might miss some interesting conjectures. It is also well known that the addition of a few key lemmas may make automated theorem proving much easier. The same is true for finite model searching [3]. We think that it can be beneficial to generate some simple formulas systematically and then test them for theoremhood.

About 10 years ago, Wos [4] listed "theorem finding" as a basic research problem in automated reasoning. What he had in mind is adopting some criteria that will help us select interesting theorems from those deduced by a theorem prover. The subject of this short paper is to find some simple conjectures which are *likely* to be theorems. They are tested in a set of small finite models of the underlying theory. If they pass the test, they become interesting conjectures.

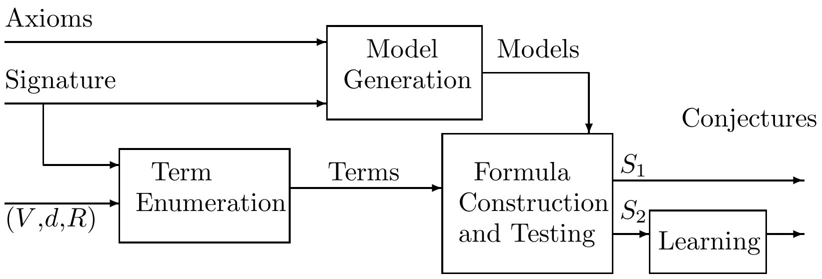
2 The Approach

Our approach is essentially generate-and-test. For a given theory which consists of a signature and a set of axioms, we need to do the following:

1. Generate a set of finite models.
2. Generate a set of closed well-formed formulas.
3. Test the truth of each formula in the models. Depending on the results, we divide the set of formulas into 3 subsets: S_0 (containing formulas which are false in every model), S_1 (containing formulas which are true in every model) and S_2 (containing the other formulas).

* Partially supported by the Natural Science Foundation of China (No. 69703014).

4. Formulas in S_0 are discarded. Each formula in S_1 is a conjecture. As for formulas in S_2, we may discover possible relationships between them using machine learning techniques.
5. (Optional) Try to prove the conjectures using a theorem prover or to refute them using a model generator.



The overall process can be described by the above diagram. In the process, we can use existing provers like OTTER to prove conjectures, and use existing finite model generators (such as FINDER, SEM or MACE) to find the models and counterexamples. In addition, we need tools for enumerating terms, generating formulas and so on. In what follows, we shall give some details about them.

Obviously, for nontrivial theories, there can be too many formulas (even restricted to some length). We choose to generate some simple terms first, and then construct formulas of certain forms. To produce terms, we implemented a procedure called *STerms()* [5]. It has 3 parameters: V (a set of variables), d (a positive integer) and R (a set of rewrite rules).

The terms generated by the procedure will contain no variables other than those listed in V, and they are different from each other up to variable renaming[1]. The number of function symbol occurrences in each term will not exceed d. In addition, if R is not empty, the terms will be irreducible with respect to R. Currently our tool is not able to perform AC rewriting. When a theory has AC functions, we use McCune's equational prover EQP to reduce the number of generated terms. (Any other AC rewriting tool could also be used.)

Given a set of terms, the formulas are constructed in the following way:

```
for each pair of terms t1 and t2 {
  find the variables appearing in the terms;
  formulate different combinations of quantifiers;
  for each combination PREN
    output the formula "PREN (t1 = t2)" ;
}
```

[1] This choice is arguable. On one hand, it prevents the generation of redundant conjectures such as $\forall x\,[f(x,x) = x]$ and $\forall y\,[f(y,y) = y]$. On the other hand, we will not be able to find the commutative law, since $f(x,y)$ and $f(y,x)$ are not generated simultaneously. However, the user can add any interesting terms manually.

As an option, one can choose to generate equations only (in which all variables are universally quantified). Otherwise, we may get too many theorems, some of which are not so interesting. For example, suppose we have the theorem $\forall x \forall y\, [\phi(x,y) = \varphi(x,y)]$, then we also have $\exists x \exists y\, [\phi(x,y) = \varphi(x,y)]$, and $\forall x \exists y\, [\phi(x,y) = \varphi(x,y)]$, and so on.

Step 3 is realized in a straightforward way. For every (closed) formula generated in Step 2, we check its truth in each model.

For the formulas which are true in some models but false in the others (i.e., those in the subset S_2), we may find some relationships between them through inductive learning. For example, which formula implies another formula. We implemented a method which is similar to the least-commitment search algorithm (Section 18.5 of [1]). Its functionality is to find a minimal set of formulas whose conjunction implies a given formula. (There may be more than one minimal sets.)

The tools are implemented in C (on SPARCstations). They communicate through files. For example, when we study group theory, we work with the files `Group1.term`, `Group1.mod` and so on.

3 Experiments

Example 1. Quasigroups. The QG5 theory [3] has the following 4 axioms:

$$\begin{array}{ll} f(x,y) = f(x,z) \rightarrow y = z & \quad f(x,x) = x \\ f(x,z) = f(y,z) \rightarrow x = y & \quad f(f(f(y,x),y),y) = x \end{array}$$

We first generate three models, of sizes 5, 7 and 8, respectively. Then we enumerate some terms. Suppose we choose $V = \{x,\, y\}$, $d = 3$, and R consists of the rule $f(x,x) \Rightarrow x$. Then there will be 23 terms, as given below.

```
x                    f(x,y)               f(x,f(x,y))
f(x,f(y,x))          f(f(x,y),x)          f(f(x,y),y)
f(x,f(x,f(x,y)))     f(x,f(x,f(y,x)))     f(x,f(y,f(x,y)))
f(x,f(y,f(y,x)))     f(x,f(f(x,y),x))     f(x,f(f(x,y),y))
f(x,f(f(y,x),x))     f(x,f(f(y,x),y))     f(f(x,y),f(y,x))
f(f(x,f(x,y)),x)     f(f(x,f(x,y)),y)     f(f(x,f(y,x)),x)
f(f(x,f(y,x)),y)     f(f(f(x,y),x),x)     f(f(f(x,y),x),y)
f(f(f(x,y),y),x)     f(f(f(x,y),y),y)
```

Only 12 pairs of them are equal in all the 3 models. All of the 12 conjectures turn out to be theorems. They can be proved by any state-of-the-art prover. The simplest of these theorems include the last axiom and the following three:

$$f(f(y,f(x,y)),y) = x \qquad f(y,f(f(x,y),y)) = x \qquad f(x,f(y,x)) = f(f(x,y),x)$$

The first two are generally believed to be helpful in solving the QG5 problem[3].

We have also experimented with other theories, like group theory, rings, boolean algebras and so on. Some results are given in Table 1. The meanings of the parameters are as follows:

Theory	Models		Terms				Conjectures			
	m	n	v	d	r	t	q	k	p	f
Group	6	5	2	2	0	23	253	16	16	0
GrpH	6	5	2	2	17	29	406	5	3	2
Ring	6	7	2	2	4	42	861	23	19	4
Ring	6	8	3	2	9	39	741	8	8	0
Bool	8	3	1	2	10	17	136	37	37	0
Bool	8	3	2	2	19	24	276	23	23	0

Table 1. Experiments with Some Abstract Algebras

m:	the maximum size of the models;
n:	the number of models;
v:	the size of V;
r:	the size of R;
t:	the number of generated terms;
q:	the number of equational formulas;
k:	the number of formulated conjectures;
p:	the number of true conjectures;
f:	the number of false conjectures.

We have, $q = t(t-1)/2$ and $k = p+f$. GrpH refers to group theory extended with commutators. For this theory, one conjecture (i.e., $h(f(x,y),x) = h(g(x),y)$) is falsified by a 12-element model; another one (i.e., $h(h(x,y),x) = h(x,h(y,x))$) is also false, but we failed to find a finite counterexample (of size up to 20).

All conjectures are equational in the first example and in Table 1. The next example is about implicational conjectures in group theory.

Example 2. Group theory. Suppose we want to know under what conditions a group is Abelian. We set $V = \{x, y\}$ and $d = 2$. Using a set of 11 groups (whose sizes are between 4 and 8), we found more than 10 equations, each of which implies commutativity. Some of the equations are given below.

$$x^2 = e; \quad x^{-1} = x; \quad x^3 = x.$$

All conjectures are proved to be theorems. But in an earlier attempt, we used 5 groups (of sizes 4, 5 and 6). And one conjecture is wrong:

$$\forall x\,(x^3 = x^{-1}) \;\rightarrow\; \forall x \forall y\,(xy = yx).$$

4 Concluding Remarks

Conjecture formulation as outlined above combines term rewriting, theorem proving, model generation and machine learning. As we said in the Introduction, it helps to produce the targets of theorem proving (i.e., the theorems to be proved), and it helps to prove certain theorems or to find certain models quickly. This may also be considered as an interesting application of finite models.

In many cases, key lemmas and good theorems are very simple formulas. We believe that the model-based conjecture searching process is very useful, especially when studying a new theory or when trying to prove a new theorem. It is also feasible, since we have good model generators now. Usually we do not need too many models, and the models do not have to be very big, to distinguish between "good" (i.e., theorems) and "bad" (i.e., false conjectures).

Of course, it is inevitable that a few conjectures turn out to be false. But the number of such conjectures can be reduced, if more models are used. And some of them may lead to interesting discoveries. In the case of modes, for example, even though a conjecture like $\forall x \forall y\, [\, f(x,y) = x\,]$ is not valid, it can hold under certain conditions [6].

Useful lemmas can also be generated by a resolution style, forward chaining program. DELTA [2], for example, generates unit clauses by applying UR-resolution, and adds them to the original formula. Such tools generate conclusions (rather than conjectures) automatically, many of which are quite complicated. But most resolution-based programs do not deal with quantified formulas directly. Our approach may be considered as complementary to forward chaining.

Combinatorial explosion is a major obstacle, although some measures have been taken (such as exploiting rewrite rules). In the future, we shall study techniques for the automatic discovery and selection of more complex, more interesting conjectures.

Acknowledgment

One referee helped to determine the theoremhood of some conjectures.

References

1. Russell, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1995.
2. Schumann, J.M.Ph., "DELTA – A bottom-up preprocessor for top-down theorem provers," *Proc. CADE-12*, 774–777, 1994.
3. Slaney, J. *et al.* "Automated reasoning and exhaustive search: Quasigroup existence problems," *Computers and Math. with Applications* 29(2): 115–132, 1995.
4. Wos, L., *Automated Reasoning: 33 Basic Research Problems,* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1988.
5. Zhang, J., *The Generation and Applications of Finite Models*, PhD thesis, Institute of Software, Chinese Academy of Sciences, Beijing, China, 1994.
6. Zhang, J., "Constructing finite algebras with FALCON," *J. Automated Reasoning* 17(1): 1–22, 1996.

Embedding Programming Languages in Theorem Provers

Tobias Nipkow

Technische Universität München
Institut für Informatik
`http://www.in.tum.de/~nipkow/`

The theory of programming languages is one of the core areas of computer science offering a wealth of models and methods. Yet the complexity of most real programming languages means that a complete formalization of their semantics is only of limited use unless it is supported by mechanical means for reasoning about the formalization. This line of research started in earnest with the seminal paper by Gordon [1] who *defined* the semantics of a simple `while`-language in the HOL system and *derived* Hoare logic from the semantics. Since then, an ever growing number of more and more sophisticated programming languages have been embedded in theorem provers. This talk surveys some of the important developments in this area before concentrating on a specific instance, *Bali*. Bali (`http://isabelle.in.tum.de/Bali/`) is an embedding of a subset of Java in Isabelle/HOL. So far, the following aspects have been covered:

1. Bali: operational semantics, type system, and a proof of type safety [2,3].
2. Bali Virtual Machine: operational semantics, type system, and a specification of the bytecode verifier together with a proof that the bytecode verifier specification guarantees type safety [4].

Current developments include a Hoare logic for Bali, a verified compiler and a verified bytecode verifier.

We hope to demonstrate that theorem proving is a key technology for formalizing and reasoning about real programming languages.

References

1. M. Gordon. Mechanizing programming logics in higher order logic. In G. Birtwistle and P. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*. Springer-Verlag, 1989.
2. T. Nipkow and D. v. Oheimb. Java$_{\ell ight}$ is type-safe — definitely. In *Proc. 25th ACM Symp. Principles of Programming Languages*, pages 161–170. ACM Press, 1998.
3. D. v. Oheimb and T. Nipkow. Machine-checking the Java specification: Proving type-safety. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1999.
4. C. Pusch. Proving the soundness of a Java bytecode verifier specification in Isabelle/HOL. In W. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *Lect. Notes in Comp. Sci.*, pages 89–103. Springer-Verlag, 1999.

Extensional Higher-Order Paramodulation and RUE-Resolution

Christoph Benzmüller

Fachbereich Informatik, Universität des Saarlandes
chris@ags.uni-sb.de

Abstract. This paper presents two approaches to primitive equality treatment in higher-order (HO) automated theorem proving: a calculus $\mathcal{EP}$ adapting traditional first-order (FO) paramodulation [RW69] , and a calculus $\mathcal{ERUE}$ adapting FO RUE-Resolution [Dig79] to classical type theory, i.e., HO logic based on Church's simply typed λ-calculus. $\mathcal{EP}$ and $\mathcal{ERUE}$ extend the extensional HO resolution approach $\mathcal{ER}$ [BK98a]. In order to reach Henkin completeness without the need for additional extensionality axioms both calculi employ new, positive extensionality rules analogously to the respective negative ones provided by $\mathcal{ER}$ that operate on unification constraints. As the extensionality rules have an intrinsic and unavoidable difference-reducing character the HO paramodulation approach loses its pure term-rewriting character. On the other hand examples demonstrate that the extensionality rules harmonise quite well with the difference-reducing HO RUE-resolution idea.

1 Introduction

Higher-Order (HO) Theorem Proving based on the resolution method has been first examined by Andrews [And71] and Huet [Hue72]. Whereas the former avoids unification the latter generally delays the computation of unifiers and instead adds unification constraints to the clauses in order to tackle the undecidability problem of HO unification. More recent papers concentrate on the adaption of sorts [Koh94] or theory unification [Wol93] to HO logic. Common to all these approaches is that they do not sufficiently solve the extensionality problem in HO automated theorem proving, i.e., all these approaches require the extensionality axioms to be added into the search space in order to reach Henkin completeness (which is the most general notion of semantics that allows complete calculi [Hen50]). This leads to a search space explosion that is awkward to manage in practice. A solution to the problem is provided by the extensional HO resolution calculus $\mathcal{ER}$ [BK98a]. This approach avoids the extensionality axioms and instead extends the syntactical (pre-)unification process by additional extensionality rules. These new rules allow for recursive calls during the (pre-) unification process to the overall refutation search whenever pure syntactical HO unification is too weak to show that two terms can be equalised modulo the extensionality principles. $\mathcal{ER}$ has been implemented in Leo [BK98b] and case studies have demonstrated its suitability, especially for reasoning about sets.

There are many possibilities to improve the extensional HO resolution approach and the probably most promising one concerns the treatment of equality. $\mathcal{ER}$ assumes that equality is defined by the Leibniz principle (two things are equal iff they have the same properties) or by any other valid definition principle, and thus provides no support for primitive equality. But a primitive equality treatment seems to be more appropriate as it avoids the many flexible literals introduced when using defined equality, which unfortunately increase the amount of blind search with $\mathcal{ER}$'s primitive substitution rule *Prim*. Therefore we adapt two well known first-order (FO) approaches to primitive equality: the paramodulation approach [RW69] (the basis of many successful refinements such as the superposition approach) and the RUE-resolution approach [Dig79] (a generalisation of E-resolution [Dar68]). The main goal thereby is to preserve Henkin completeness. We will show that therefore positive extensionality rules are needed (which operate on positive equation literals) as in contrast to FO logic single positive equations can be contradictory by themselves in HO logic.[1]

This paper summarises the Chapt. 6, 7, and 8 of [Ben99] and because of lack of space the preliminaries and the formal proofs can only be sketched here.

The preliminaries are concisely presented in Sect. 2 and calculus $\mathcal{ER}$ is reviewed in 3. Section 4 discusses interesting aspects on primitive and defined equality, before the extensional HO paramodulation calculus $\mathcal{EP}$ and the extensional HO RUE-resolution approach $\mathcal{ERUE}$ are discussed in 5 and 6. Both approaches are briefly compared by examples in 7 and the conclusion is presented in 8.

2 Higher-Order (HO) Logic

We consider a HO logic based on Church's simply typed λ-calculus [Chu40] and choose $BT := \{\iota, o\}$ as *base types*, where ι denotes the set of individuals and o the set of truth values. *Functional types* are inductively defined over BT. A *signature* Σ ($\Sigma^=$) contains for each type an infinite set of variables and constants and provides the *logical connectives* $\neg_{o\rightarrow o}$, $\vee_{o\rightarrow o\rightarrow o}$, and $\Pi_{(\alpha\rightarrow o)\rightarrow o}$ (additionally $=^\alpha \;:=\; =_{\alpha\rightarrow\alpha\rightarrow o}$) for every type α. The set of all Σ-terms (closed Σ-terms) of type α is denoted by wff_α (cwff_α). Variables are printed as upper-case (e.g. X_α), constants as lower-case letters (e.g. c_α) and arbitrary terms appear as bold capital letters (e.g. $\mathbf{T}_\alpha$). If the type of a symbol is uniquely determined by the given context we omit it. We abbreviate function applications by $h_{\alpha_1\rightarrow\cdots\rightarrow\alpha_n\rightarrow\beta}\,\overline{\mathbf{U}^n_{\alpha_n}}$, which stands for $(\cdots(h_{\alpha_1\rightarrow\cdots\rightarrow\alpha_n\rightarrow\beta}\,\mathbf{U}^1_{\alpha_1})\cdots\mathbf{U}^n_{\alpha_n})$. For α-, β-, η-, $\beta\eta$-*conversion* and the definition of $\beta\eta$- and *head-normal form* (*hnf*) for a term $\mathbf{T}$ we refer to [Bar84] as well as for the definition of *free* variables, *closed* formulas, and *substitutions*. *Unification* and sets of *partial bindings* $\mathcal{AB}^h_\gamma$ are well explained in [SG89]. An example for a *pre-clause*, i.e., not in proper clause normal form, consisting of a positive literal, a negative literal, and a special negative equation literal (also called *unification constraint*) is $\mathcal{C} : [\neg(P_{\iota\rightarrow o}\,\mathbf{T}_\iota)]^T \vee [h_{\overline{\gamma}\rightarrow o}\,\overline{Y^n_{\gamma_n}}]^F \vee [Q_{\iota\rightarrow\iota}\,a_\iota = Y_{\iota\rightarrow\iota}\,b_\iota]^F$. The corresponding *proper clause*, i.e., properly normalised, is $\mathcal{C}' : [P\;\mathbf{T}]^F \vee [h\;\overline{Y^n}]^F \vee [Q\;a = Y\;b]^F$. The unification constraint in $\mathcal{C}$ and $\mathcal{C}'$ is called a *flex-flex* pair as both unification terms have *flexible* heads. A clause is called *empty*, denoted by $\square$, if it consists

[1] Consider, e.g. the positive literal $[a_o = \neg a_o]^T$ or $[G\;X = f]^T$ (resulting from the following formulation of Cantor's theorem: $\neg\exists G_{\iota\rightarrow\iota\rightarrow o}.\,\forall P_{\iota\rightarrow o}.\,\exists X_\iota.\,G\;X = P$).

only of *flex-flex* unification constraints. A clause $\mathcal{C}_1$ generalises a clause $\mathcal{C}_2$, iff there is a substitution σ, such that the $\beta\eta$-normal form of $\sigma(\mathcal{C}_1)$ is an α-variant of the $\beta\eta$-normal form of $\mathcal{C}_2$.

A calculus R provides a set of rules $\{r_n|\ 0 < n \leq i\}$ defined on clauses. We write $\Phi \vdash^{r_n} \mathcal{C}$ ($\mathcal{C}' \vdash^{r_n} \mathcal{C}$) iff clause $\mathcal{C}$ is the result of an one step application of rule $r_n \in R$ to premise clauses $\mathcal{C}'_i \in \Phi$ (to $\mathcal{C}'$ respectively). Multiple step derivations in calculus R are abbreviated by $\Phi_1 \vdash_R \Phi_k$ (or $\mathcal{C}_1 \vdash_R \mathcal{C}_k$).

A *standard model* for $\mathcal{HOL}$ provides a fixed set $\mathcal{D}_\iota$ of individuals, and a set $\mathcal{D}_o := \{\top, \bot\}$ of truth values. The domains for functional types are defined inductively: $\mathcal{D}_{\alpha\to\beta}$ is the set of all functions $f\colon \mathcal{D}_\alpha \to \mathcal{D}_\beta$. *Henkin models* only require that $\mathcal{D}_{\alpha\to\beta}$ has enough members that any well-formed formula can be evaluated. Thus, the generalisation to Henkin models restricts the set of valid formulae sufficiently, such that complete calculi are possible. In Henkin and standard semantics Leibniz equality ($\dot{=}^\alpha := \lambda X_\alpha\text{.}\ \lambda Y_\alpha\text{.}\ \forall P_{\alpha\to o}\text{.}\ PX \Rightarrow PY$) denotes the intuitive equality relation and the functional extensionality principles ($\forall M_{\alpha\to\beta}\text{.}\ \forall N_{\alpha\to\beta}\text{.}\ (\forall X\text{.}\ (MX) = (NX)) \Leftrightarrow (M = N)$) as well as the Boolean extensionality principle ($\forall P_o\text{.}\ \forall Q_o\text{.}\ (P = Q) \Leftrightarrow (P \Leftrightarrow Q)$) are valid (see [Ben99,BK97]). *Satisfiability* and *validity* ($\mathcal{M} \models \mathbf{F}$ or $\mathcal{M} \models \Phi$) of a formula $\mathbf{F}$ or set of formulas Φ in a model $\mathcal{M}$ is defined as usual.

The completeness proofs employ the abstract consistency method of [BK97]& [Ben99] which extends Andrews' HO adaptation [And71] of Smullyan's approach [Smu63] to Henkin semantics. Here we only mention the two main aspects:

Definition 1 (*Acc* for Henkin Models). *Let Σ be a signature and Γ_Σ a class of sets of Σ-sentences. If the following conditions (all but ∇_e^*) hold for all $\mathbf{A}, \mathbf{B} \in cwff_o$, $\mathbf{F}, \mathbf{G} \in cwff_{\alpha\to\beta}$, and $\Phi \in \Gamma_\Sigma$, then we call Γ_Σ an* **abstract consistency class for Henkin models with primitive equality**, *abbreviated by Acc$^=$ (resp.* **abstract consistency class for Henkin models**, *abbreviated by Acc).*

Saturated $\Phi \cup \{\mathbf{A}\} \in \Gamma_\Sigma$ *or* $\Phi \cup \{\neg\mathbf{A}\} \in \Gamma_\Sigma$.
∇_c *If* $\mathbf{A}$ *is atomic, then* $\mathbf{A} \notin \Phi$ *or* $\neg\mathbf{A} \notin \Phi$.
$\nabla_\neg$ *If* $\neg\neg\mathbf{A} \in \Phi$, *then* $\Phi \cup \{\mathbf{A}\} \in \Gamma_\Sigma$.
∇_f *If* $\mathbf{A} \in \Phi$ *and* $\mathbf{B}$ *is the $\beta\eta$-normal form of* $\mathbf{A}$, *then* $\Phi \cup \{\mathbf{B}\} \in \Gamma_\Sigma$.
$\nabla_\vee$ *If* $\mathbf{A} \vee \mathbf{B} \in \Phi$, *then* $\Phi \cup \{\mathbf{A}\} \in \Gamma_\Sigma$ *or* $\Phi \cup \{\mathbf{B}\} \in \Gamma_\Sigma$.
$\nabla_\wedge$ *If* $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$, *then* $\Phi \cup \{\neg\mathbf{A}, \neg\mathbf{B}\} \in \Gamma_\Sigma$.
$\nabla_\forall$ *If* $\Pi^\alpha\mathbf{F} \in \Phi$, *then* $\Phi \cup \{\mathbf{F}\ \mathbf{W}\} \in \Gamma_\Sigma$ *for each* $\mathbf{W} \in cwff_\alpha$.
$\nabla_\exists$ *If* $\neg\Pi^\alpha\mathbf{F} \in \Phi$, *then* $\Phi \cup \{\neg(\mathbf{F}\ w)\} \in \Gamma_\Sigma$ *for any new constant* $w \in \Sigma_\alpha$.
∇_b *If* $\neg(\mathbf{A} \dot{=}^o \mathbf{B}) \in \Phi$, *then* $\Phi \cup \{\mathbf{A}, \neg\mathbf{B}\} \in \Gamma_\Sigma$ *or* $\Phi \cup \{\neg\mathbf{A}, \mathbf{B}\} \in \Gamma_\Sigma$.
∇_q *If* $\neg(\mathbf{F} \dot{=}^{\alpha\to\beta} \mathbf{G}) \in \Phi$, *then* $\Phi \cup \{\neg(\mathbf{F}\ w \dot{=}^\beta \mathbf{G}\ w)\} \in \Gamma_\Sigma$ *for any new constant* $w \in \Sigma_\alpha$.
∇_e^r $\neg(\mathbf{A}_\alpha = \mathbf{A}) \notin \Phi$. ∇_e^s *If* $\mathbf{F}[\mathbf{A}]_p \in \Phi$ *and* $\mathbf{A} = \mathbf{B} \in \Phi$, *then* $\Phi \cup \{\mathbf{F}[\mathbf{B}]_p\} \in \Gamma_\Sigma$.[2]

Theorem 1 (Henkin Model Existence). *Let Φ be a set of closed Σ-formulas ($\Sigma^=$-formulas), Γ_Σ ($\Gamma_{\Sigma^=}$) be an Acc (Acc$^=$) and $\Phi \in \Gamma_\Sigma$. There exists a Henkin model $\mathcal{M}$, such that $\mathcal{M} \models \Phi$.*

$$\frac{\mathbf{C} \vee [\mathbf{A} \vee \mathbf{B}]^T}{\mathbf{C} \vee [\mathbf{A}]^T \vee [\mathbf{B}]^T}\ \vee^T \qquad \frac{\mathbf{C} \vee [\mathbf{A} \vee \mathbf{B}]^F}{\mathbf{C} \vee [\mathbf{A}]^F}\ \vee^F_l \qquad \frac{\mathbf{C} \vee [\mathbf{A} \vee \mathbf{B}]^F}{\mathbf{C} \vee [\mathbf{B}]^F}\ \vee^F_r$$

$$\frac{\mathbf{C} \vee [\neg \mathbf{A}]^T}{\mathbf{C} \vee [\mathbf{A}]^F}\ \neg^T \qquad \frac{\mathbf{C} \vee [\neg \mathbf{A}]^F}{\mathbf{C} \vee [\mathbf{A}]^T}\ \neg^F \qquad \frac{\mathbf{C} \vee [\Pi^\gamma \mathbf{A}]^T \quad X_\gamma \text{ new variable}}{\mathbf{C} \vee [\mathbf{A}\ X]^T}\ \Pi^T$$

$$\frac{\mathbf{C} \vee [\Pi^\gamma \mathbf{A}]^F \quad \mathrm{sk}_\gamma \text{ is a Skolem term for this clause}}{\mathbf{C} \vee [\mathbf{A}\ \mathrm{sk}_\gamma]^F}\ \Pi^F$$

Fig. 1. Clause Normalisation Calculus $\mathcal{CNF}$

$\boxed{\text{Clause Normalisation:}}$

$$\frac{\mathcal{D} \quad \mathcal{C} \in \mathcal{CNF}(\mathcal{D})}{\mathcal{C}}\ Cnf$$

$\boxed{\text{Resolution:}}$ (defined on proper clauses only)

$$\frac{[\mathbf{A}]^\alpha \vee \mathbf{C} \quad [\mathbf{B}]^\beta \vee \mathbf{D} \quad \alpha \neq \beta}{\mathbf{C} \vee \mathbf{D} \vee [\mathbf{A} = \mathbf{B}]^F}\ Res \qquad \frac{[\mathbf{A}]^\alpha \vee [\mathbf{B}]^\alpha \vee \mathbf{C} \quad \alpha \in \{T, F\}}{[\mathbf{A}]^\alpha \vee \mathbf{C} \vee [\mathbf{A} = \mathbf{B}]^F}\ Fac$$

$$\frac{[Q_\gamma\ \overline{\mathbf{U}^k}]^\alpha \vee \mathbf{C} \quad \mathbf{P} \in \mathcal{AB}_\gamma^{\{\neg,\vee\}\cup\{\Pi^\beta | \beta \in \mathcal{T}^k\}},\ \alpha \in \{T, F\}}{[Q_\gamma\ \overline{\mathbf{U}^k}]^\alpha \vee \mathbf{C} \vee [Q = \mathbf{P}]^F}\ Prim$$

$\boxed{\text{Extensional (Pre-)Unification:}}$

$$\frac{\mathbf{C} \vee [\mathbf{M}_{\gamma\to\beta} = \mathbf{N}_{\gamma\to\beta}]^F \quad s_\gamma \text{ Skolem term for this clause}}{\mathbf{C} \vee [\mathbf{M}\ s = \mathbf{N}\ s]^F}\ Func$$

$$\frac{\mathbf{C} \vee [\mathbf{A}_{\gamma\to\beta}\ \mathbf{C}_\gamma = \mathbf{B}_{\gamma\to\beta}\ \mathbf{D}_\gamma]^F}{\mathbf{C} \vee [\mathbf{A} = \mathbf{B}]^F \vee [\mathbf{C} = \mathbf{D}]^F}\ Dec$$

$$\frac{\mathbf{C} \vee [\mathbf{A} = \mathbf{A}]^F}{\mathbf{C}}\ Triv \qquad \frac{\mathbf{C} \vee [X = \mathbf{A}]^F \quad X \text{ does not occur in } \mathbf{A}}{\mathbf{C}_{\{\mathbf{A}/X\}}}\ Subst$$

$$\frac{\mathbf{C} \vee [F_\gamma\ \overline{\mathbf{U}^n} = h\ \overline{\mathbf{V}^m}]^F \quad \mathbf{G} \in \mathcal{AB}^h_\gamma}{\mathbf{C} \vee [F\ \overline{\mathbf{U}^n} = h\ \overline{\mathbf{V}^m} \vee [F = \mathbf{G}]^F]^F}\ FlexRigid$$

$$\frac{\mathbf{C} \vee [\mathbf{M}_o = \mathbf{N}_o]^F}{\mathbf{C} \vee [\mathbf{M}_o \Leftrightarrow \mathbf{N}_o]^F}\ Equiv \qquad \frac{\mathbf{C} \vee [\mathbf{M}_\gamma = \mathbf{N}_\gamma]^F}{\mathbf{C} \vee [\forall P_{\gamma\to o}\text{.}\ P\ \mathbf{M} \Rightarrow P\ \mathbf{N}]^F}\ Leib$$

$$\left(\frac{\mathbf{C} \vee [F_{\overline{\gamma^n}\to\gamma}\ \overline{\mathbf{U}^n} = H_{\overline{\delta^m}\to\gamma}\ \overline{\mathbf{V}^m}]^F \quad \mathbf{G} \in \mathcal{AB}^h_{\overline{\gamma^n}\to\gamma} \text{ for a constant } h_\tau}{\mathbf{C} \vee [F\ \overline{\mathbf{U}^n} = H\ \overline{\mathbf{V}^m}]^F \vee [F = \mathbf{G}]^F}\ FlexFlex\right)$$

$\mathcal{AB}^h_\gamma$ specifies the set of partial bindings of type γ for head h as defined in [SG89]

Fig. 2. Extensional HO Resolution Calculus $\mathcal{ER}$

3 $\mathcal{ER}$: Extensional HO Resolution

Figure 1 presents calculus $\mathcal{CNF} := \{\vee^T, \vee_l^F, \vee_r^F, \neg^T, \neg^F, \Pi^T, \Pi^F\}$ for clause normalisation. These rules are defined on (pre-)clauses and are known to preserve validity or satisfiability with respect to standard semantics.[3]

The syntactical unification rules (cf. Fig. 2) provided by $\mathcal{ER}$ which operate on unification constraints are $\mathcal{UNI} := \{Func, Dec, Triv, Subst,\ FlexRigid\}$. These rules realise a sound and complete approach to HO pre-unification. Note the double role of extensionality rule *Func*: on the one hand this rule works as a syntactical unification rule and subsumes the α- and η-rule as, e.g. presented in [BK98a]; on the other hand *Func* applies the functional extensionality principle if none of the two terms is a λ-abstraction. Apart from rule *Func*, $\mathcal{ER}$ provides the extensionality rules *Equiv* and *Leib* (cf. Fig. 2). The former applies the Boolean extensionality principle and the latter simply replaces a negative unification constraint (encoded as a negative equation) by a negative Leibniz equation. The extensionality rules operate on unification constraints only and do in contrast to the respective axioms not introduce flexible heads into the search space.

The main proof search is performed by the resolution rule *Res* and the factorisation rule *Fac*. It is furthermore well known for HO resolution, that the primitive substitution rule *Prim* is needed to ensure Henkin completeness.

For the calculi presented in this paper we assume that the result of each rule application is transformed into hnf[4], where the hnf of unification constraints is defined special and requires both unification terms to be reduced to hnf. A set of formulas Φ is refutable in calculus R, iff there is a derivation $\Delta : \Phi_{cl} \vdash_R \Box$, where $\Phi_{cl} := \{[\mathbf{F}']^T | \mathbf{F}' \text{ hnf of } \mathbf{F} \in \Phi\}$ is the clause-set obtained from Φ by simple pre-clausification. More details on $\mathcal{ER}$ are provided by [BK98a,Ben99].

Whereas completeness of $\mathcal{ER}$ has already been analysed in [BK98a] this paper (and [Ben99]) presents an alternative completeness proof for a slightly extended version of $\mathcal{ER}$ (this version, e.g. employs the instantiation guessing *FlexFlex*-rule). The new proof is motivated as follows: (i) it eases the proof of the lifting lemma and avoids the quite complicated notion of clause isomorphisms as used in [BK98a,Koh94], (ii) it can be reused to show the completeness for calculi $\mathcal{EP}$ and $\mathcal{ERUE}$ as well, (iii) it prepares the analysis of non-normal form resolution calculi, and (iv) it emphasises interesting aspects on rule *FlexFlex*, unification, and clause normalisation wrt. $\mathcal{ER}$, $\mathcal{EP}$, and $\mathcal{ERUE}$.

One such interesting aspect is that different to Huet [Hue72] eager unification is essential within our approach. This is illustrated by the argumentations for ∇_b and ∇_q in the completeness proofs (cf. [Ben99,BK98a]) as well as the examples presented in Sec. 7 or [Ben99]. However, we claim that rule *FlexFlex* can still be delayed until the end of a refutation, i.e., *FlexFlex* can be completely avoided.

The author has not been able to prove the latter claim yet. And thus the completeness proofs for $\mathcal{ER}$ (and $\mathcal{EP}$, $\mathcal{ERUE}$) still depends on the *FlexFlex*-rule.

[2] $\mathbf{A}$ does not contain free variables.
[3] For Skolemisation we employ Miller's sound HO correction [Mil83].
[4] One may also $\beta\eta$-normal form here.

We now sketch the main results on $\mathcal{ER}$ as discussed in detail in [Ben99].

Definition 2 (Extensional HO Resolution). *We define three calculi:*
$\mathcal{ER}$:= $\{Cnf, Res, Fac, Prim\} \cup \mathcal{UNI} \cup \{Equiv, Leib\}$ *employs all rules (except* FlexFlex*) displayed in Fig. 2.*
$\mathcal{ER}_f$:= $\mathcal{ER} \cup \{FlexFlex\}$ *uses full HO unification instead of pre-unification.*
$\mathcal{ER}_{fc}$:= $(\mathcal{ER}_f \backslash \{Cnf\}) \cup \mathcal{CNF}$ *employs unfolded and stepwise clause normalisation instead of exhaustive normalisations with rule* Cnf*.*
These calculi treat equality as a defined notion only (e.g. by Leibniz equality) and primitive equations are not allowed in problem formulations. Although unification constraints are encoded as negative equation literals, no rule but the unification rules are allowed to operate on them.

Theorem 2 (Soundness). *The calculi $\mathcal{ER}$, $\mathcal{ER}_f$, and $\mathcal{ER}_{fc}$ are Henkin-sound (H-sound).*

Proof. Preservation of validity or satisfiability with respect to Henkin semantics is proven analogously to the standard FO argumentation. For Skolemisation (employed in rule Π^F and *Func*) we use Miller's sound HO version [Mil83]. Soundness of the extensionality rules *Equiv, Func*, and *Leib* is obvious as they simply apply the valid extensionality principles.

Lemma 1 (Lifting of $\mathcal{ER}_{fc}$). *Let Φ be clause set, $\mathcal{D}_1$ a clause, and σ a substitution. If $\sigma(\Phi) \vdash_{\mathcal{ER}_{fc}} \mathcal{D}_1$, then $\Phi \vdash_{\mathcal{ER}_{fc}} \mathcal{D}_2$ for a clause $\mathcal{D}_2$ generalising $\mathcal{D}_1$.*

Proof. One can easily show that each instantiated derivation can be reused on the uninstantiated level as well. In blocking situations caused by free variables at literal head position or at unification term head position, either rule *Prim* or rule *FlexFlex* can be employed in connection with rule *Subst* to introduce the missing term structure. The rather unintuitive clause isomorphisms of [BK98a] or [Koh94] are thereby avoided.

Theorem 3 (Completeness). *Calculus $\mathcal{ER}_{fc}$ is Henkin complete.*

Proof. Analogously to the proof in [BK98a] we show that the set of closed formulas that are not refutable in $\mathcal{ER}_{fc}$ (i.e., $\Gamma_\Sigma := \{\Phi \subseteq cwff_o | \Phi_{cl} \not\vdash_{\mathcal{ER}_{fc}} \Box\}$) is a saturated abstract consistency class for Henkin models (cf. Def. 1). This entails Henkin completeness for $\mathcal{ER}_{fc}$ by Thm. 1.

Lemma 2 (Theorem Equivalence). *The calculi $\mathcal{ER}_{fc}$ and $\mathcal{ER}_f$ are theorem equivalent, i.e., for each clause set Φ holds that $\Phi \vdash_{\mathcal{ER}_{fc}} \Box$ iff $\Phi \vdash_{\mathcal{ER}_f} \Box$.*

Proof. We can even prove a more general property: For each proper clause $\mathcal{C}$ holds $\Phi \vdash_{\mathcal{ER}_{fc}} \mathcal{C}$ implies $\Phi \vdash_{\mathcal{ER}_f} \mathcal{C}$. The proof idea is to show that the unfolded clause normalisations can be grouped together and then replaced by rule *Cnf*.

Question 1 (Theorem Equivalence). The author claims that the calculi $\mathcal{ER}$ and $\mathcal{ER}_{fc}$ (or $\mathcal{ER}_f$) are theorem equivalent. A formal proof has not been carried out yet. Some evidence is given by the case studies carried out with the LEO-prover [BK98b] and the direct completeness proof for $\mathcal{ER}$ in [BK98a].

4 Primitive Equality

Treating equality as a defined notion in HO logic (e.g. by the Leibniz principle) is convenient in theory, but often inefficient and unintuitive in practical applications as many free literal heads are introduced into the search space, which increases the degree of blind search with primitive substitution rule *Prim*.[5] This is the main motivation for the two approaches to primitive equality presented in the next sections. Before we discuss these approaches in detail we point to the following interesting aspects of defined equality in HO logic:

- There are infinitely many different valid definitions of equality in HO logic.[6] For instance: Leibniz equality ($\dot{=}^{\alpha} := \lambda X_\alpha.\, \lambda Y_\alpha.\, \forall P_{\alpha\to o}.\, PX \Rightarrow PY$), Reflexivity definition[7] ($\ddot{=}^{\alpha} := \lambda X_\alpha.\, \lambda Y_\alpha.\, \forall Q_{\alpha\to\alpha\to o}.\, (\forall Z_\alpha.\, (Q\ Z\ Z)) \Rightarrow (Q\ X\ Y)$), and infinitely many artificial modifications to all valid definitions (e.g., $\dddot{=}^{\alpha} := \lambda X_\alpha.\, \lambda Y_\alpha.\, \forall P_{\alpha\to o}.\, ((a_o \vee \neg\, a_o) \wedge P\ X) \Rightarrow ((b_o \vee \neg\, b_o) \wedge P\ Y))$. The latter definition is obviously equivalent to Leibniz definition as it just adds some tautologies to the embedded formulas.
- The artificially modified definitions demonstrate, that it is generally not decidable whether a formula is a valid definition of equality (as the set of tautologies is not decidable). Hence, it is not decidable whether an input problem to one of our proof procedures contains a valid definition of equality, and we cannot simply replace all valid definitions embedded in a problem formulation by primitive equations as one might wish to.

If we are interested in Henkin completeness, we therefore have to ensure that the paramodulation and RUE-resolution approaches presented in the next sections can handle all forms of defined equality (like the underlying calculus $\mathcal{ER}$) and can additionally handle primitive equality.[8]

5 $\mathcal{EP}$: Extensional HO Paramodulation

In this section we adapt the well known FO paramodulation approach [RW69] to our HO setting and examine Henkin completeness. A straightforward adaptation of the traditional FO paramodulation rule is given by rule *Para* in Fig. 3. Analogous to the $\mathcal{ER}$ rules *Res* and *Fac*, (pre-)unification is delayed by encoding the respective unification problem (its solvability justifies the rewriting step) as

[5] This is illustrated by the examples that employ defined equality in [BK98a] and the examples that employ primitive equality in Sect. 7.

[6] For this statement we assume Henkin or standard semantics as underlying semantical notion. In weaker semantics things get even more complicated as, e.g., Leibniz equality does not necessary denote the intended equality relation. For a detailed discussion see [Ben99,BK97].

[7] As presented in Andrews textbook [And86], p. 155.

[8] The author admits, that in practice one is mainly interested in finding proofs rather than in the theoretical notion of Henkin completeness. Anyhow, our motivation in this paper is to clarify the theoretical properties of our approaches.

$$\frac{[\mathbf{A}[\mathbf{T}_\gamma]]^\alpha \vee C \quad [\mathbf{L}_\gamma = \mathbf{R}_\gamma]^T \vee D}{[\mathbf{A}[\mathbf{R}]]^\alpha \vee C \vee D \vee [\mathbf{T} = \mathbf{L}]^F}\ Para \qquad \frac{[\mathbf{A}]^\alpha \vee C \quad [\mathbf{L}_\gamma = \mathbf{R}_\gamma]^T \vee D}{[P_{\gamma\to o}\ \mathbf{R}]^\alpha \vee C \vee D \vee [\mathbf{A} = P\ \mathbf{L}]^F}\ Para'$$

We implicitly assume the symmetric application of $[\mathbf{L}_\gamma = \mathbf{R}_\gamma]^T$.
$\mathbf{T}$ (in *Para*) does not contain free variables which are bound outside of $\mathbf{T}$.

Fig. 3. Adapted Paramodulation Rule and a HO specific reformulation

a unification constraint. Rule *Para'* is an elegant HO specific reformulation[9] of paramodulation that has a very simple motivation: It describes the resolution step with the clause $[P\ \mathbf{L}]^F \vee [P\ \mathbf{R}]^T \vee D$, i.e., the clause obtained when replacing the primitive equation $[\mathbf{L} = \mathbf{R}]^T$ by its Leibniz definition. Note that the paramodulant of *Para'* encodes all possible single rewrite steps, all simultaneous rewrite-steps with rule *Para*, and in some sense even the left premise clause itself. This is nicely illustrated by the following example: $\mathcal{C}_1 : [p\ (f\ (f\ a))]^T$ and $\mathcal{C}_2 : [f = h]^T$, where $p_{\iota\to o}, f_{\iota\to\iota}, h_{\iota\to\iota}$ are constants. Applying rule *Para'* to $\mathcal{C}_1$ and $\mathcal{C}_2$ from left to right leads to $\mathcal{C}_3 : [P_{(\iota\to\iota)\to\iota}\ h]^T \vee [p\ (f\ (f\ a)) = P_{(\iota\to\iota)\to\iota}\ f]^F$. Eager unification computes the following four solutions for P, which can be back-propagated to literal $[P\ h]^T$ with rule *Subst*:

$[\lambda Z_{\iota\to\iota}\text{.}\ p\ (f\ (f\ a))/P]$ the pure imitation solution encodes $\mathcal{C}_1$ itself.
$[\lambda Z_{\iota\to\iota}\text{.}\ p\ (Z\ (f\ a))/P]$ encodes the rewriting of the first f ($[p\ (h\ (f\ a))]^T$).
$[\lambda Z_{\iota\to\iota}\text{.}\ p\ (f\ (Z\ a))/P]$ encodes the rewriting of the second f ($[p\ (f\ (h\ a))]^T$).
$[\lambda Z_{\iota\to\iota}\text{.}\ p\ (Z\ (Z\ a))/P]$ encodes the simult. rewr. of both f ($[p\ (h\ (h\ a))]^T$).

Rule *Para'* introduces flexible literal heads into the search space such that rule *Prim* becomes applicable. Thus, a probably suitable heuristics in practice is to avoid all primitive substitution steps on flexible heads generated by rule *Para'*.

Note that reflexivity resolution[10] and paramodulation into unification constraints[11] are derivable in our approach and can thus be avoided.

[9] This rule was first suggested by Michael Kohlhase.

[10] In FO a reflexivity resolution rule is needed to refute negative equation literals $[\mathbf{T}_1 = \mathbf{T}_2]^F$ if $\mathbf{T}_1$ and $\mathbf{T}_2$ are unifiable. As such literals are automatically treated as unification constraints reflexivity resolution is not needed in our approach.

[11] Let $\mathcal{C}_1 : C \vee [\mathbf{A}[\mathbf{T}] = \mathbf{B}]^F$ and $\mathcal{C}_2 : [\mathbf{L} = \mathbf{R}]^T \vee D$. The rewriting step $Para(\mathcal{C}_1, \mathcal{C}_2) : \mathcal{C}_3 : C \vee D \vee [\mathbf{A}[\mathbf{R}] = \mathbf{B}]^F \vee [\mathbf{L} = \mathbf{T}]^F$ can be replaced by derivation $Leib(\mathcal{C}_1) : \mathcal{C}_4 : [p\ \mathbf{A}[\mathbf{T}]]^T \vee C,\ \mathcal{C}_5 : [p\ \mathbf{B}]^F \vee C;\ Para(\mathcal{C}_4, \mathcal{C}_2) : \mathcal{C}_6 : [p\ \mathbf{A}[\mathbf{R}]]^T \vee C \vee D \vee [\mathbf{L} = \mathbf{T}]^F;\ Res(\mathcal{C}_6, \mathcal{C}_5), Fac, Triv : \mathcal{C}_7 : C \vee D \vee [p\ \mathbf{A}[\mathbf{R}] = p\ \mathbf{B}]^F \vee [\mathbf{L} = \mathbf{T}]^F;\ Dec(\mathcal{C}_7) : \mathcal{C}_3$. Notational remark: $Res(\mathcal{C}_6, \mathcal{C}_5), Fac, Triv$ describes the application of rule *Res* to $\mathcal{C}_6$ and $\mathcal{C}_5$, followed by applications of *Fac* and *Triv* to the subsequent results.

In the following discussion we will use the traditional paramodulation rule *Para* only.[12] As *Para'* is obviously more general than *Para* we obtain analogous completeness results if we employ *Para'* instead.

Definition 3 (Simple HO Paramodulation). $\mathcal{EP}_{naive} := \mathcal{ER} \cup \{Para\}$ *extends the extensional HO resolution approach by rule* Para. *Primitive equations in input problems are no longer expanded by Leibniz definition.* Para *operates on proper clause only and omits paramodulation into unification constraints.*

Whereas soundness of rule *Para* can be shown analogously to the FO case, it turns out that our simple HO paramodulation approach is incomplete:

Theorem 4 (Incompleteness). *Calculus $\mathcal{EP}_{naive}$ is Henkin incomplete.*

Proof. Consider the following counterexamples: $\mathbf{E}_1^{Para}$**:** $\neg\exists X_o\text{.}\ (X = \neg X)$, i.e., the negation operator is fix-point free, which is obviously the case in Henkin semantics. Negation and clause normalisation leads to clause $\mathcal{C}_1 :\ [a = \neg a]^T$, where a_o is a new Skolem constant. The only rule that is applicable is self-paramodulation at positions $\langle 1\rangle$, $\langle 2\rangle$, and $\langle\rangle$, leading to the following clauses (including the symmetric rewrite steps):

$$
\begin{array}{llll}
Para(\mathcal{C}_1,\mathcal{C}_1)\ at\ \langle 1\rangle : \mathcal{C}_2 : & [a = \neg a]^T \vee [\neg a = a]^F, & \mathcal{C}_3 : & [\neg a = \neg a]^T \vee [a = a]^F \\
Para(\mathcal{C}_1,\mathcal{C}_1)\ at\ \langle 2\rangle : \mathcal{C}_4 : & [a = \neg a]^T \vee [a = \neg a]^F, & \mathcal{C}_5 : & [a = a]^T \vee [\neg a = \neg a]^F \\
Para(\mathcal{C}_1,\mathcal{C}_1)\ at\ \langle\rangle : \ \mathcal{C}_6 : & [a]^T \vee [\neg a = (a = \neg a)]^F, & \mathcal{C}_7 : & [a]^F \vee [a = (a = \neg a)]^F
\end{array}
$$

A case distinction on the possible denotations $\{\top, \bot\}$ for a shows that all clauses are tautologous, such that no refutation is possible in $\mathcal{EP}_{naive}$. Additional examples are discussed in [Ben99], e.g. $\mathbf{E}_2^{Para}$**:** $[G\ X = p]^T$, which stems from a simple version of cantor's theorem $\neg\exists G_{\iota\to\iota\to o}\text{.}\ \forall P_{\iota\to o}\text{.}\ \exists X_\iota\text{.}\ G\ X = P$, or example $\mathbf{E}_3^{Para}$**:** $[M = \lambda X_o\text{.}\ \bot]^T$, which stems from $\exists M_{o\to o}\text{.}\ M \neq \emptyset$.

The problem is that in HO logic even single positive equation literals can be contradictory. And the incompleteness is caused as the extensionality principles are now also needed to refute such positive equation literals.[13] Hence, we add the positive counterparts *Func'* and *Equiv'* (cf. Fig. 4) to the already present negative extensionality rules *Func* and *Equiv*. The completeness proof and the examples show that a positive counterpart for rule *Leib* can be avoided.

Definition 4 (Extensional HO Paramodulation). *Analogously to the extensional HO resolution case we define the calculi $\mathcal{EP} :=\ \mathcal{ER} \cup \{Para, Equiv',$ $Func'\}$, $\mathcal{EP}_f :=\ \mathcal{EP} \cup \{FlexFlex\}$, and $\mathcal{EP}_{fc} :=\ (\mathcal{EP}_f \backslash \{Cnf\}) \cup \mathcal{CNF}$.*

Theorem 5 (Soundness). *The calculi $\mathcal{EP}$, $\mathcal{EP}_f$, and $\mathcal{EP}_{fc}$ are H-sound.*

[12] It has been pointed out by a unknown referee of this paper that rule *Para'* already captures full functional extensionality and should therefore be preferred over *Para*. Example $\mathbf{E}_1^{func}$ discussed in Sec. 10.6 of [Ben99] illustrates that this is generally not true.

[13] In contrast to $\mathcal{EP}$, the underlying calculus $\mathcal{ER}$ does not allow positive equation literals as the equality symbol is only used to encode unification constraints. Therefore the pure extensional HO resolution approach $\mathcal{ER}$ does not require a positive extensionality treatment.

$$\frac{\mathbf{C} \lor [\mathbf{M}_o = \mathbf{N}_o]^T}{\mathbf{C} \lor [\mathbf{M}_o \Leftrightarrow \mathbf{N}_o]^T}\ Equiv' \qquad \frac{\mathbf{C} \lor [\mathbf{M}_{\gamma\to\beta} = \mathbf{N}_{\gamma\to\beta}]^T \quad X \text{ new variable}}{\mathbf{C} \lor [\mathbf{M}\ X_\gamma = \mathbf{N}\ X_\gamma]^T}\ Func'$$

Fig. 4. Positive Extensionality Rules

Proof. Soundness of rule *Para* with respect to Henkin semantics can be proven analogously to the FO case and soundness of *Equiv'* and *Func'* is obvious, as they simply apply the extensionality principles, which are valid in Henkin semantics.

Lemma 3 (Lifting of $\mathcal{EP}_{fc}$). *Let Φ be a clause set, $\mathcal{D}_1$ a clause, and σ a substitution. If $\sigma(\Phi) \vdash_{\mathcal{ER}_{fc}} \mathcal{D}_1$, then $\Phi \vdash_{\mathcal{EP}_{fc}} \mathcal{D}_2$ for a clause $\mathcal{D}_2$ generalising $\mathcal{D}_1$.*

Proof. Analogous to Lemma 1. The additional rules do not cause any problems.

The main completeness theorem 6 for $\mathcal{EP}_{fc}$ below is proven analogously to Thm. 3, i.e., we employ the model existence theorem for Henkin models with primitive equality (cf. Thm. 1). As primitive equality is involved, we additionally have to ensure the abstract consistency properties ∇_e^r and ∇_e^s (cf. Def. 1), i.e., the reflexivity and substitutivity property of primitive equality. Whereas the reflexivity property is trivially met, we employ the following admissible[14] — and moreover even weakly derivable (i.e., modulo clause normalisation and lifting) — paramodulation rule to verify the substitutivity property.

Definition 5 (Generalised Paramodulation). *The generalised paramodulation rule* GPara *is defined as follows:*

$$\frac{[\mathbf{T}[\mathbf{A}_\beta]]^\alpha \lor C \quad [\mathbf{A}_\beta = \mathbf{B}_\beta]^T}{[\mathbf{T}[\mathbf{B}]]^\alpha \lor C}\ GPara$$

This rule extends Para *as it can be applied to non-proper clauses and it restricts* Para *as it can only be applied in special clause contexts, e.g. the second clause has to be a unit clause.* GPara *is especially designed to verify the substitutivity property of primitive equality ∇_e^s in the main completeness theorem 6.*

Weak derivability (which obviously implies admissibility) of *GPara* is shown with the help of the following weakly derivable generalised resolution rules.

Definition 6 (Generalised Resolution). *The generalised resolution rules $GRes_1$, $GRes_2$, and $GRes_3$ are defined as follows (for all rules we assume $\alpha, \beta \in \{T, F\}$ with $\alpha \neq \beta$, and for $GRes_2$ we assume that $\overline{Y^n} \notin \mathbf{free}(\mathbf{A})$):*

[14] Rule r is called admissible (derivable) in R, iff adding rule r to calculus R does not increase the set of refutable formulas (iff each application of rule r can be replaced by an alternative derivation in calculus R).

$$\frac{[\mathbf{A}_{\overline{\gamma}\to o}\ \overline{\mathbf{T}^n_\gamma}]^\alpha \vee C \quad [\mathbf{A}_{\overline{\gamma}\to o}\ \overline{X^n_\gamma}]^\beta \vee D}{(C \vee D)_{[\overline{T^n}/\overline{X^n}]}}\ GRes_1 \qquad \frac{[\mathbf{A}_\gamma\ \overline{Y^n}]^\alpha \vee C \quad [X_\gamma\ \overline{\mathbf{T}^n}]^\beta \vee D}{(C \vee D)_{[\mathbf{A}/X, \overline{\mathbf{T}^n}/\overline{Y^n}]}}\ GRes_2$$

$$\frac{[\mathbf{A}_\gamma\ \overline{\mathbf{T}^n}]^\alpha \vee C \quad [X_\gamma\ \overline{Y^n}]^\beta \vee D}{(C \vee D)_{[\mathbf{A}/X, \overline{\mathbf{T}^n}/\overline{Y^n}]}}\ GRes_3$$

These rules extend Res *as they can be applied to non-proper clauses, and they restrict* Res *as they are only defined for special clause contexts. The rules are designed just strong enough to prove weak derivability of* GPara.

Lemma 4 (Weak Derivability of $GRes_{1,2,3}$**).** *Let* $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ *be clauses and* $r \in \{GRes_1, GRes_2, GRes_3\}$. *If* $\{\mathcal{C}_1, \mathcal{C}_2\} \vdash^r \mathcal{C}_3 \vdash_{\mathcal{CNF}} \mathcal{C}_4$ *for a proper clause* $\mathcal{C}_4$, *then* $\{\mathcal{C}_1, \mathcal{C}_2\} \vdash_{\mathcal{EP}_{fc}} \mathcal{C}_5$ *for a clause* $\mathcal{C}_5$ *which generalises* $\mathcal{C}_4$.

Proof. The proof is by induction on the number of logical connectives in the resolution literals. It employs generalised (and weakly derivable) versions of the factorisation rule *Fac* and primitive substitution rule *Prim* (see [Ben99]), which are not presented here because lack of space. $GRes_2$ and $GRes_3$ are needed to prove weak derivability for $GRes_1$. As the rules *Para, Equiv', Func'* are not employed in the proof, this lemma analogously holds for calculus $\mathcal{ER}_{fc}$.

Lemma 5 (Weak Derivability of *GPara***).** *Let* $\mathcal{C}_1 : [\mathbf{T}[\mathbf{A}]_p]^\alpha \vee D_1, \mathcal{C}_2 : [\mathbf{A} = \mathbf{B}]^T, \mathcal{C}_3 : [\mathbf{T}[\mathbf{B}]_p]^\alpha \vee D_1$ *be clauses. If* $\Delta : \{\mathcal{C}_1, \mathcal{C}_2\} \vdash^{GPara} \mathcal{C}_3 \vdash_{\mathcal{CNF}} \mathcal{C}_4$ *for a proper clause* $\mathcal{C}_4$, *then* $\{\mathcal{C}_1, \mathcal{C}_2\} \vdash_{\mathcal{EP}_{fc}} \mathcal{C}_5$ *for a clause* $\mathcal{C}_5$ *generalising* $\mathcal{C}_4$.

Proof. The proof is by induction on the length of Δ and employs the (weakly derivable) generalised resolution rule $GRes_1$ and the standard paramodulation rule *Para* in the quite complicated base case.

Theorem 6 (Completeness). *Calculus* $\mathcal{EP}_{fc}$ *is Henkin complete.*

Proof. Let Γ_Σ be the set of closed Σ-formulas that cannot be refuted with calculus $\mathcal{EP}_{fc}$ (i.e., $\Gamma_\Sigma := \{\Phi \subseteq cwff_o | \Phi_{cl} \not\vdash_{\mathcal{EP}_{fc}} \Box\}$). We show that Γ_Σ is a saturated abstract consistency class for Henkin models with primitive equality (cf. Def. 1). This entails completeness by the model existence theorem for Henkin models with primitive equality (cf. Thm. 1).

First we have to verify that Γ_Σ validates the abstract consistency properties ∇_c, $\nabla_\neg$, ∇_β, $\nabla_\vee$, $\nabla_\wedge$, $\nabla_\forall$, $\nabla_\exists$, ∇_b, ∇_q and that Γ_Σ is saturated. In all of these cases the proofs are identical to the corresponding argumentations in Thm. 3.

Thus, all we need to ensure is the validity of the additional abstract consistency properties ∇_e^r and ∇_e^s for primitive equality:
(∇_e^r) We have that $[\mathbf{A} =^\alpha \mathbf{A}]^F \vdash^{Triv} \Box$, and thus $\neg(\mathbf{A} =^\alpha \mathbf{A})$ cannot be in Φ.
(∇_e^s) Analogously to the cases in Sec. 3 we show the contrapositive of the assertion, and thus we assume that there is derivation $\Delta : \Phi_{cl} \cup \{[\mathbf{F}[\mathbf{B}]]^T\} \vdash_{\mathcal{EP}_{fc}} \Box$. Now consider the following $\mathcal{EP}_{fc}$-derivation: $\Delta' : \Phi_{cl} \cup \{[\mathbf{F}[\mathbf{A}]]^T, [\mathbf{A} = \mathbf{B}]^T\} \vdash^{GPara} \Phi_{cl} \cup \{[\mathbf{F}[\mathbf{A}]]^T, [\mathbf{A} = \mathbf{B}]^T, [\mathbf{F}[\mathbf{B}]]^T\} \vdash_{\mathcal{EP}_{fc}} \Box$. By Lemma 5 *GPara* is weakly derivable (hence admissible) for calculus $\mathcal{EP}_{fc}$, such that there is a $\mathcal{EP}_{fc}$-derivation $\Delta'' : \Phi_{cl} \cup \{[\mathbf{F}[\mathbf{A}]]^T, [\mathbf{A} = \mathbf{B}]^T\} \vdash_{\mathcal{EP}_{fc}} \Phi_{cl} \cup \{[\mathbf{F}[\mathbf{A}]]^T, [\mathbf{A} = \mathbf{B}]^T, [\mathbf{F}[\mathbf{B}]]^T\} \vdash_{\mathcal{EP}_{fc}} \Box$ which completes the proof.

Lemma 6 (Theorem Equivalence). *$\mathcal{EP}_{fc}$ and $\mathcal{EP}_f$ are theorem equivalent.*

Proof. Analogous to Lemma 2. The additional rules do not cause any problems.

Question 2 (Theorem Equivalence). The author claims that the calculi $\mathcal{EP}$ and $\mathcal{EP}_{fc}$ (or $\mathcal{EP}_f$) are theorem equivalent. The formal proof will most likely be analogous to the one for question 1.

6 $\mathcal{ERUE}$: Extensional HO RUE-Resolution

In this section we will adapt the **R**esolution by **U**nification and **E**quality approach [Dig79] to our higher-order setting. The key idea is to allow the resolution and factorisation rules also to operate on unification constraints (which is forbidden in $\mathcal{ER}$ and $\mathcal{EP}$). This implements the main ideas of FO RUE-resolution directly in our higher-order calculus. More precisely our approach allows to compute partial E-unifiers with respect to a specified theory E by resolution on unification constraints within the calculus itself (if we assume that E is specified in form of an available set of unitary or even conditional equations in clause form). This is due to the fact that the extensional higher-order resolution approach already realises a test calculus for general higher-order E-pre-unification (or higher-order E-unification in case we also add the rule *FlexFlex*). Furthermore, each partial E-(pre-)unifier can be applied to a clause with rule *Subst*, and, like in the traditional FO RUE-resolution approach, the non-solved unification constraints are encoded as (still open) unification constraints, i.e., negative equations, within the particular clauses.

Definition 7 (Extensional HO RUE-Resolution). *We now allow the factorisation rule* Fac *and resolution rule* Res *to operate also on unification constraints and define the calculi $\mathcal{ERUE} := \mathcal{ER} \cup \{Equiv', Func'\}$, $\mathcal{ERUE}_f := \mathcal{ERUE} \cup \{FlexFlex\}$, and $\mathcal{ERUE}_{fc} := (\mathcal{ERUE}_f \backslash \{Cnf\}) \cup \mathcal{CNF}$.*

Theorem 7 (Soundness). *The calculi $\mathcal{ERUE}_{fc}$, $\mathcal{ERUE}_f$, and $\mathcal{ERUE}$ are H-sound.*

Proof. Unification constraints are encoded as negative literals, such that soundness of the extended resolution and factorisation rules with respect to Henkin semantics is obvious.

Lemma 7 (Lifting of $\mathcal{ERUE}_{fc}$). *Let Φ be a clause set, $\mathcal{D}_1$ a clause, and σ a substitution. If $\sigma(\Phi) \vdash_{\mathcal{ER}_{fc}} \mathcal{D}_1$, then $\Phi \vdash_{\mathcal{ERUE}_{fc}} \mathcal{D}_2$ for a clause $\mathcal{D}_2$ generalising $\mathcal{D}_1$.*

Proof. Analogous to Lemmata 1 and 3.

Within the main completeness proof we proceed analogously to previous section and employ the generalised paramodulation rule *GPara* to verify the crucial substitutivity property ∇_e^s. Thus, we need to show that *GPara* is admissible in calculus $\mathcal{ERUE}_{fc}$. Note that in Lemma 5 we were even able to show a weak derivability property of rule *GPara* for calculus $\mathcal{EP}_{fc}$. Whereas *GPara* is not weakly derivability for calculus $\mathcal{ERUE}_{fc}$, we can still prove admissibility of this rule here. As in Lemma 5, we employ the generalised resolution rules which are weakly derivable in $\mathcal{ERUE}_{fc}$ as well.

Lemma 8 (Weak Derivability of $GRes_{1,2,3}$). *Let $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ be clauses and $r \in \{GRes_1, GRes_2, GRes_3\}$. If $\{\mathcal{C}_1, \mathcal{C}_2\} \vdash^r \mathcal{C}_3 \vdash_{\mathcal{CNF}} \mathcal{C}_4$ for a proper clause $\mathcal{C}_4$, then $\{\mathcal{C}_1, \mathcal{C}_2\} \vdash_{\mathcal{ERUE}_{fc}} \mathcal{C}_5$ for a clause $\mathcal{C}_5$ which generalises $\mathcal{C}_4$.*

Proof. Analogous to Lemma 4.

Lemma 9 (Admissibility of $GPara$). *Let Φ be a clause set, such that $\Delta : \Phi \vdash^{GPara} \Phi' \vdash_{\mathcal{ERUE}_{fc}} \Box$, then there exists a refutation $\Phi \vdash_{\mathcal{ERUE}_{fc}} \Box$.*

Proof. The proof is (analogous to Lemma 5) by induction on the length of Δ and employs the weakly derivable generalised resolution rule $GRes_1$. The applications of rule *Para* in the proof of Lemma 5 are replaced by corresponding derivations employing resolution and factorisation on unification constraints. The latter causes the loss of the weak derivability property.

Theorem 8 (Completeness). *Calculus $\mathcal{ERUE}_{fc}$ is Henkin complete.*

Proof. Analogously to Lemma 6 we show that the set of closed Σ-formulas which cannot be refuted by the calculus $\mathcal{ERUE}_{fc}$ (i.e., $\Gamma_\Sigma := \{\Phi \subseteq cwff_o | \Phi_{cl} \nvdash_{\mathcal{ERUE}_{fc}} \Box\}$) is a saturated abstract consistency class for Henkin models with primitive equality (cf. Def. 1). This entails the assertion by Thm. 1.

The proof is analogous to Lemma 6. Even the abstract consistency properties ∇_e^r and ∇_e^s are proven analogously by employing the generalised paramodulation rule *GPara*, which is by Lemma 9 admissible in $\mathcal{ERUE}_{fc}$.

Lemma 10 (Theorem Equiv.). *$\mathcal{ERUE}_{fc}$ and $\mathcal{ERUE}_f$ are theorem equivalent.*

Proof. Analogous to Lemma 2. The additional or modified rules do not cause any problems.

Question 3 (Theorem Equivalence). The author claims that the calculi $\mathcal{ERUE}$ and $\mathcal{ERUE}_{fc}$ (or $\mathcal{ERUE}_f$) are theorem equivalent. A formal proof will most likely be analogous to questions 1 and 2.

7 Examples

The first (trivial FO) example illustrates the main ideas of $\mathcal{EP}$ and $\mathcal{ERUE}$: $a_\iota \in m_{\iota\to o} \wedge a = b \Rightarrow b \in m$. Sets are encoded as characteristic functions and $\in := \lambda X_\alpha, M_{\alpha\to o}.\, M\ X$, such that the negated problem normalises to: $\mathcal{C}_1 : [m\ a]^T$, $\mathcal{C}_2 : [a = b]^T$, $\mathcal{C}_3 : [m\ b]^F$. An obvious term-rewriting refutation in $\mathcal{EP}$: $Para(\mathcal{C}_1, \mathcal{C}_2)$, $Triv : \mathcal{C}_4 : [m\ b]^T$; $Res(\mathcal{C}_3, \mathcal{C}_4), Triv : \Box$.[15] A difference-reducing refutation in $\mathcal{ERUE}$: $Res(\mathcal{C}_1, \mathcal{C}_3) : \mathcal{C}_4 : [m\ a = m\ b]^F$; $Dec(\mathcal{C}_4), Triv : \mathcal{C}_5 : [a = b]^F$; $Res(\mathcal{C}_2, \mathcal{C}_5), Triv : \Box$.

We now examine the examples mentioned in Thm. 4 in calculus $\mathcal{EP}$: $\mathbf{E}_2^{Para}$: $[(G\ X_\iota) = p_{\iota\to o}]^T$ (Cantor's theorem) $Func'(\mathbf{E}_2^{Para}), Equiv'$: C_1 : $[G\ X\ Y_\iota]^F \vee$

[15] Notation (as already used before): $Res(\mathcal{C}_6, \mathcal{C}_5), Fac$ describes a paramodulation step between $\mathcal{C}_6$ and $\mathcal{C}_5$ followed by factorisation of the resulting clause. $Prim(\mathcal{C}_1 | \mathcal{C}_2)$ denotes the parallel application of rule *Prim* to $\mathcal{C}_j$ and $\mathcal{C}_k$.

$[p\ Y_\iota]^T$, $\mathcal{C}_2 : [G\ X\ Y_\iota]^T \vee [p\ Y_\iota]^F$; $\mathit{Prim}(\mathcal{C}_1|\mathcal{C}_2), \mathit{Subst} : \mathcal{C}_3 : [G'\ X\ Y]^T \vee [p\ Y]^T$, $\mathcal{C}_4 : [G''\ X\ Y]^F \vee [p\ Y]^F$; $\mathit{Fac}(\mathcal{C}_3|\mathcal{C}_4), \mathcal{UNI} : \mathcal{C}_5 : [p\ Y]^T$, $\mathcal{C}_6 : [p\ Y]^F$; $\mathit{Res}(\mathcal{C}_5, \mathcal{C}_6), \mathcal{UNI} : \mathcal{C}_7 : \square$. $\mathbf{E}_1^{Para}$ and $\mathbf{E}_3^{Para}$ can be proven analogously. The key idea is to employ the positive extensionality rules first. As paramodulation rule is not employed, these proofs are obviously also possible in $\mathcal{ERUE}$.

Example $\mathbf{E}_2^{set}$ focuses on reasoning about sets: $(\{X|\ \mathit{odd}\ X \wedge \mathit{num}\ X\} = \{X|\ \neg\ \mathit{ev}\ X \wedge \mathit{num}\ X\}) \Rightarrow (2^{\{X|\ \mathit{odd}\ X \wedge X > 100 \wedge \mathit{num}\ X\}} = 2^{\{X|\ \neg\ \mathit{ev}\ X \wedge X > 100 \wedge \mathit{num}\ X\}})$, where the powerset-operator is defined by $\lambda N_{\alpha\to o}\text{.}\ \lambda M_{\alpha\to o}\text{.}\ \forall X_\alpha\text{.}\ \mathbf{M}\ X \Rightarrow \mathbf{N}\ X$. $\mathcal{CNF}(\mathbf{E}_2^{set}), \mathit{Func}, \mathit{Func}' : \mathcal{C}_1 : [(\mathit{odd}\ X \wedge \mathit{num}\ X) = (\neg\ \mathit{ev}\ X \wedge \mathit{num}\ X)]^T$ and $\mathcal{C}_2 : [(\forall X\text{.}\ n\ X \Rightarrow ((\mathit{odd}\ X \wedge X > 100) \wedge \mathit{num}\ X)) = (\forall X\text{.}\ n\ X \Rightarrow ((\neg\ \mathit{ev}\ X \wedge X > 100) \wedge \mathit{num}\ X))]^F$ where n is a Skolem constant. The reader may check that an application of rule *Para* does not lead to a successful refutation here as the terms in the powerset description do unfortunately not have the *right structure*. Instead of following the term-rewriting idea we have to proceed with difference-reduction and a recursive call to the overall refutation search from within the unification process: $\mathit{Dec}(\mathcal{C}_2), \mathit{Triv}, \mathit{Func}, \mathit{Dec}, \mathit{Triv} : \mathcal{C}_3 : [((\mathit{odd}\ s \wedge s > 100) \wedge \mathit{num}\ s) = ((\neg\ \mathit{ev}\ s \wedge s > 100) \wedge \mathit{num}\ s)]^F$; $\mathit{Equiv}(\mathcal{C}_3), \mathcal{CNF}, \mathit{Fac}, \mathcal{UNI}: \mathcal{C}_4 : [\mathit{odd}\ s]^T \vee [\mathit{ev}\ s]^F$, $\mathcal{C}_5 : [s > 100]^T$, $\mathcal{C}_6 : [\mathit{num}\ s]^T$, $\mathcal{C}_7 : [\mathit{odd}\ s]^F \vee [s > 100]^F \vee [\mathit{num}\ s]^F \vee [\mathit{ev}\ s]^T$; $\mathit{Equiv}'(\mathcal{C}_1), \mathcal{CNF}, \mathit{Fac}, \mathcal{UNI}: \mathcal{C}_8 : [\mathit{odd}\ X]^F \vee [\mathit{num}\ X]^F \vee [\mathit{ev}\ X]^F$, $\mathcal{C}_9 : [\mathit{odd}\ X]^T \vee [\mathit{num}\ X]^F \vee [\mathit{ev}\ X]^T$. The rest of the refutation is a straightforward resolution proof on $\mathcal{C}_4 - \mathcal{C}_9$. It is easy to check that an elegant term-rewriting proof is only possible if we put the succedent of $\mathbf{E}_2^{set}$ in the *right order*: $2^{\{X|\ (\mathit{odd}\ X \wedge \mathit{num}\ X) \wedge X > 100\}} = 2^{\{X|\ (\neg\ \mathit{ev}\ X \wedge \mathit{num}\ X) \wedge X > 100\}}$. Thus this example nicely illustrates the unavoidable mixed term-reducing and difference-reducing character of extensional higher-order paramodulation.

On the other hand a very interesting goal directed proof is possible within the RUE-resolution approach $\mathcal{ERUE}$ by immediately resolving between $\mathcal{C}_1$ and the unification constraint $\mathcal{C}_2$ and subsequently employing syntactical unification in connection with recursive calls to the overall refutation process (with the extensionality rules) when syntactical unification is blocked.

[Ben99] provides a more detailed discussion of these and additional examples.

8 Conclusion

We presented the two approaches $\mathcal{EP}$ and $\mathcal{ERUE}$ for extensional higher-order paramodulation and RUE-resolution which extend the extensional higher-order resolution approach $\mathcal{ER}$ [BK98a] by a primitive equality treatment. All three approaches avoid the extensionality axioms and employ more goal directed extensionality rules instead. An interesting difference to Huet's original constraint resolution approach [Hue72] is that eager (pre-)unification becomes essential and cannot be generally delayed if an extensionality treatment is required.

Henkin completeness has been proven for the slightly extended (by the additional rule *FlexFlex*) approaches $\mathcal{ER}_f$, $\mathcal{EP}_f$ and $\mathcal{ERUE}_f$. The claim that rule *FlexFlex* is admissible in them has not been proven yet. All three approaches can be implemented in a higher-order set of support approach as presented in [Ben99]. [Ben99] also presents some first ideas how the enormous search space of the introduced approaches can be further restricted in practice, e.g. by introducing redundancy methods.

It has been motivated that some problems cannot be solved in the paramodulation approach $\mathcal{EP}$ by following the term-rewriting idea only, as they unavoidably require the application of the difference-reducing extensionality rules. In contrast to $\mathcal{EP}$ the difference-reducing calculus $\mathcal{ERUE}$ seems to harmonise quite well with the difference-reducing extensionality rules (or axioms), and thus this paper concludes with the question: Can HO adaptations of term-rewriting approaches be as successful as in FO, if one is interested in Henkin completeness and extensionality, e.g., when reasoning about sets, where sets are encoded as characteristic functions? Further work will be to examine this aspect with the help of the LEO-system [BK98b] and to investigate the open questions of this paper.

Acknowledgements. The work reported here was funded by the Deutsche Forschungsgemeinschaft under grant HOTEL. The author is grateful to M. Kohlhase, F. Pfenning, P. Andrews, V. Sorge and S. Autexier for stimulating discussions and support.

References

And71. P. B. Andrews. Resolution in type theory. *JSL*, 36(3):414–432, 1971.
And86. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press, 1986.
Bar84. H. P. Barendregt. *The Lambda-Calculus: Its Syntax and Semantics.* North-Holland, 2nd edition 1984.
Ben99. C. Benzmüller. Equality and Extensionality in Automated Higher-Order Theorem Proving. PhD thesis, FB 14, Universität des Saarlandes, 1999.
BK97. C. Benzmüller and M. Kohlhase. Model existence for higher-order logic. Seki-Report SR-97-09, FB 14, Universität des Saarlandes, 1997, submitted to JSL.
BK98a. C. Benzmüller and M. Kohlhase. Extensional higher-order resolution. In Kirchner and Kirchner [KK98], pages 56–72.
BK98b. C. Benzmüller and M. Kohlhase. LEO — a higher-order theorem prover. In Kirchner and Kirchner [KK98], pages 139–144.
Chu40. A. Church. A formulation of the simple theory of types. *JSL*, 5:56–68, 1940.
Dar68. J. L. Darlington. Automatic theorem proving with equality substitutions and mathematical induction. *Machine Intelligence*, 3:113–130, 1968.
Dig79. V. J. Digricoli. Resolution by unification and equality. In W. H. Joyner, editor, *Proc. of the 4th Workshop on Automated Deduction*, Austin, 1979.
Hen50. L. Henkin. Completeness in the theory of types. *JSL*, 15(2):81–91, 1950.
Hue72. G. P. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic.* PhD thesis, Case Western Reserve University, 1972.
KK98. C. Kirchner and H. Kirchner, editors. *Proc. of the 15th Conference on Automated Deduction*, number 1421 in LNAI, Springer, 1998.
Koh94. M. Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle.* PhD thesis, Universität des Saarlandes, 1994.
Mil83. D. Miller. *Proofs in Higher-Order Logic.* PhD thesis, Carnegie-Mellon University, 1983.
RW69. G. A. Robinson and L. Wos. Paramodulation and TP in first order theories with equality. *Machine Intelligence*, 4:135–150, 1969.
SG89. W. Snyder and J. H. Gallier. Higher-Order Unification Revisited: Complete Sets of Transformations. *Journal of Symbolic Computation*, 8:101–140, 1989.
Smu63. Raymond M. Smullyan. A unifying principle for quantification theory. Proc. Nat. Acad Sciences, 49:828–832, 1963.
Wol93. D. Wolfram. *The Clausal Theory of Types*. Cambridge University Press, 1993.

Automatic Generation of Proof Search Strategies for Second-Order Logic

Raul H. C. Lopes

Departamento de Informática - UFES, Caixa Postal 01-9011/29060-970 - Brazil
raulh@camburi.inf.ufes.br

Abstract. $\mathcal{P}_2$ is introduced: an algorithm for the automatic generation of proof search strategies from sets of examples of proofs. The proof search strategies are generated as sets of assertions (called *methods*) about the use of inference rules found in the examples. Sets of methods are prioritized and they can be compiled into clauses of a logic program. Proofs obtained for difficult problems in classical second-order logic are used as evidence of the adequacy of the methodology.

1 Introduction

This paper presents the algorithm $\mathcal{P}_2$, that generates assertions (called *methods*) about the uses of the inference rules of a given logic from examples of proofs. The algorithm assumes that proofs can be performed in a goal-oriented fashion, and that they can be decomposed in *proof steps*, containing a *goal* and a designation of an inference rule applied to it. Its main components are: a set of randomized algorithms for generating *methods* from proof steps, and algorithms for ordering *methods*, for matching methods with proof goals, and for establishing redundancy of *methods.* It uses examples of proofs to generate sets of methods that can be compiled into logic programs, encoding proof search strategies. However, a fundamental characteristic of the generated sets of methods is the fact they are prioritized.

$\mathcal{P}_2$ has been applied to generate provers for Intuitionistic Propositional Calculus (**IPC**), modal logics S4, and S5 (results in [10] and in [9]). In this paper, results of an experiment with classical second-order logic are shown and analyzed. In the experiment to be analyzed, a proof search strategy generated by an implementation of $\mathcal{P}_2$ was used to drive proofs in fully automatic mode for all of the problems in Fig. 1. Particularly interesting are the automatic proofs obtained for the problems 5 to 14 (taken from [3]) and for Cantor's theorem that the power set of a set S is larger than S (problem 15.)

The following properties of the generated provers will be discussed: the *soundness* of the generated provers, the *generality* of the methodology with respect to application to several logics, and the *re-usability* of the generated provers. Also discussed are the importance of assigning priorities to the *methods* that constitute a proof search strategy, and the use of a *lazy* elimination of quantifiers in proofs involving higher-order quantification as a device to avoid the explosion in the tree to be searched.

1. $\vdash \forall X \forall Y \forall Z(((X = Y) \wedge (Y = Z)) \supset (X = Z))$
2. $\vdash \forall X \forall Y((X = Y) \supset (P_1(X) \leftrightarrow P_1(Y)))$
3. $\vdash \forall X \forall Y \forall Z((X = Y) \supset (G_2(X, Z) = G_2(Y, Z)))$
4. $\vdash \forall X \forall Y((X = F_1(Y)) \supset (H_1(G_1(X)) = H_1(G_1(F_1(Y)))))$
5. $\vdash \exists A \forall X(A(X) \supset (X <= 0))$
6. $\vdash \forall P(P(a) \supset \exists A(\forall X(A(X) \supset P(X)) \wedge \exists Y A(Y)))$
7. $\vdash \forall X \forall Y \forall Z(((X < Y) \wedge (Y < Z)) \supset \exists A(\neg A(X) \wedge (A(Y) \wedge \neg A(Z))))$
8. $\vdash \forall P(P(F_1(b)) \supset \exists S \exists A(\forall X(A(X) \supset P(X)) \wedge A(F_1(S))))$
9. $\vdash ((P_2(a, b) \wedge \forall E Q_0(b, E)) \supset$
   $\exists A(\forall Y(A(Y) \supset \exists X(P_2(Y, X) \wedge Q_0(X, A))) \wedge A(a)))$
10. $\vdash \forall X \exists A \neg A(X)$
11. $\vdash \forall U \forall V((U < 0) \supset \neg(U = abs(V))) \supset$
    $\exists A(\forall Y(\neg A(abs(Y))) \wedge A(-(2)))$
12. $\vdash \forall X(\exists U(X = (2 * U)) \leftrightarrow \neg \exists V((X + 1) = (2 * V))) \supset$
    $\exists A \forall X(A(X) \leftrightarrow \neg A(X + 1))$
13. $\vdash \forall P \forall Y((\forall A((A(0) \wedge \forall X(A(X) \supset A(X + 1))) \supset A(Y)) \wedge$
    $(P(0) \wedge \forall X(P(X) \supset P(X + 1)))) \supset P(Y))$
14. $\forall X \forall Y \forall Z((F_2(X, Y) = F_2(Z, Z)) \supset (X = Y)) \vdash$
    $\forall P \forall U \forall V((\forall A((A(F_2(0, 0)) \wedge$
    $\forall X \forall Y(A(F_2(X, Y)) \supset A(F_2(F_1(X), F_1(Y))))) \supset A(F_2(U, V))) \wedge P(U)) \supset P(V))$
15. $\vdash \neg \exists G \forall F \exists J(G(J) = F)$

Fig. 1. A problem set for $\mathcal{P}_2$

2 Outline of the Methodology

$\mathcal{P}_2$ receives as input a structure representing a *set of proofs* in a given *logic*, and it generates a structure defining a *proof search strategy* for that logic. The proof search strategy is composed of *methods*. A *method* defines a recommendation for the use of an operator (for example, a recommendation for the use of an LCF tactic) of that logic, and is the analogue of the *Condition-Action* rules of rule-based systems. In that sense, $\mathcal{P}_2$ comprises an inductive generalization methodology for generating, from sets of examples, *methods*, defining conditions for the use of a set of operators, which in our case happen to be inference rules (or tactics in tactical implementations of provers). However, it will be argued that inductive generalization methodologies can and should learn from the given examples the preferences implicitly assigned to the use of some operators: when more than one operator can be applied to the same goal, the choice of one of them is an important property that should be captured by the inductive generalization procedure.

2.1 The Structure of the Provers

It is assumed that provers in general perform their proofs in a goal-oriented fashion: proving consists in solving a set of *goals*. Starting with an initial *goal*, that encodes a theorem to be proved, the prover has a set of operators that can be applied to this goal. Typically the operators implement the inference rules of the underlying logic. The application of an operator to a goal can either solve it completely, or generate a set of new goals that must be added to other unsolved goals that the prover may already have and that must be eventually all solved, if a proof is to be achieved. It is also assumed that provers are implemented, keeping a clear separation between *control* and *logic*. The generated proof search strategies are used to control the application of the inference rules of the logic in question. They do *not* interfere in the formalization of the logic and the implementation of this formalization. This issue is illustrated in Fig.2. It shows that a proof is performed through the interaction of two distinct processes: *logic*, that should implement the formalization of the underlying logic; and *driver*, that uses a *proof search strategy* to drive (control) the proof. The *logic* component is responsible for implementing the inference rules. It receives from the *driver* a recommended inference rule for a goal, and it either performs the inference returning a set of sub-goals to be solved, or fails in applying the recommended rule to the given goal.

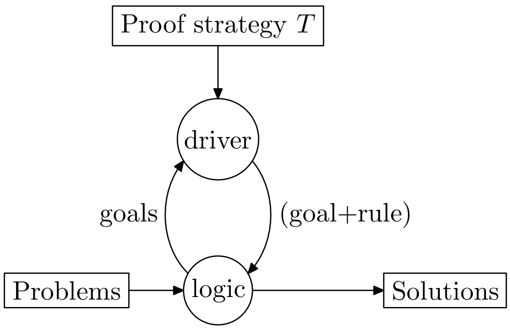


Fig. 2. Driving proofs

3 Second-Order Logic

$\mathcal{P}_2$ has been used to generate provers for several logics. In this paper, however, the examples and experiments are concentrated on the generation of proof search strategies for second-order logic. The prover used in the experiments with second-order logic, called $\mathcal{P}_2$-hol, can be decomposed in three main elements: a set of methods, generated by $\mathcal{P}_2$ and defining a proof search strategy; a *driver*; and a *logic* component. The *logic* component implements a tableau formalization of a Simple Theory of Types ([5]). The complete formalization of the logic is presented in [10]. Here, a subset of its inference rules is introduced to facilitate the discussion of $\mathcal{P}_2$.

The logic of $\mathcal{P}_2$-hol is a sub-logic of Church's Simple Theory of Types, excluding the axioms of extensionality, descriptions, and infinity. It contains two base types: o(type for propositions), and ι(type for individuals.) $(\alpha \to \beta)$ will denote the type of functions with domain type α and co-domain type β. Formulas are terms of a simply typed λ-calculus, generated from a set of atomic terms, that is itself the union of three disjoint sets: a set of logical constants, a denumerable set of free variables, and a denumerable set of parameters (non-logical constants.) This lambda calculus will be called $\lambda\!\to$.

The logic is formalized as a tableau of signed formulas. A signed formula is either a term $\mathbf{T}\,\alpha$ or $\mathbf{F}\,\alpha$ where α is a term of $\lambda\!\to$ with type o, and $\mathbf{T}$ and $\mathbf{F}$ are reserved for the signs of the formulas.

A *goal* is a tuple of prefixed signed formulas, and $\alpha \diamond \Gamma$ will represent a tuple whose first formula is α (Γ being a tuple itself.) A prefixed signed formula is a pair containing: a tuple of free variables (the prefix), and a signed formula. The prefix contains variables introduced by quantifier elimination inference rules. This is the same device used by Shankar (in [14]) to deal with quantification in intuitionistic first-order logic.

Formulas will be displayed without types and the following abbreviations will be used: $(p \supset q)$ abbreviates $((\supset p)q)$, and $\exists xP$ abbreviates $(\exists(\lambda xP))$, where: $\supset$ is a logical constant with type $(o \to o \to o)$, $\exists$ is a logical constant with type $((\iota \to o) \to o)$. Lambda abstraction is used to bind quantified variables.

The inference rules for $\supset$ and $\neg$ (that has type $(o \to o)$) are shown below.

$$\begin{array}{rl}
(\mathbf{T}\,\neg) & \text{provable}(\Pi : \mathbf{T}\,\neg\alpha \diamond \Gamma, \theta_0, \theta_1) \text{ if} \\
 & \qquad \text{provable}(\Pi : \mathbf{F}\,\alpha \diamond \Gamma, \theta_0, \theta_1) \\
(\mathbf{F}\,\neg) & \text{provable}(\Pi : \mathbf{F}\,\neg\alpha \diamond \Gamma, \theta_0, \theta_1) \text{ if} \\
 & \qquad \text{provable}(\Pi : \mathbf{T}\,\alpha \diamond \Gamma, \theta_0, \theta_1) \\
(\mathbf{T}\,\supset) & \text{provable}(\Pi : \mathbf{T}\,(\alpha \supset \beta) \diamond \Gamma, \theta_0, \theta_1) \text{ if} \\
 & \qquad \text{provable}(\Pi : \mathbf{F}\,\alpha \diamond \Gamma, \theta_0, \theta_2) \text{ and} \\
 & \qquad \text{provable}(\Pi : \mathbf{T}\,\beta \diamond \Gamma, \theta_0, \theta_3) \text{ and} \\
 & \qquad \text{unify}(\theta_2 \cup \theta_3, \theta_1) \\
(\mathbf{F}\,\supset) & \text{provable}(\Pi : \mathbf{F}\,(\alpha \supset \beta) \diamond \Gamma, \theta_0, \theta_1) \text{ if} \\
 & \qquad \text{provable}(\Pi : \mathbf{T}\,\alpha \diamond \Pi : \mathbf{F}\,\beta \diamond \Gamma, \theta_0, \theta_1)
\end{array}$$

where $\Pi : \mathbf{F}\,\alpha \diamond \Gamma$ is a tuple whose first prefixed signed formula is $\Pi : \mathbf{F}\,\alpha$. The inference rules are defined with the help of two predicates:

- unify(θ_i, θ_o), that is true if θ_o is a substitution that is a unifier for the set of pairs θ_i. (This relation can be implemented using Huet's higher-order unification algorithm introduced in [7].)
- provable$(\Gamma, \theta_0, \theta_1)$, which is true if Γ has a proof under restriction θ_0 (a restriction being a set of pairs of terms) with output unifier θ_1 (another set of pairs of terms.)

The inference rules for the quantifier $\exists$ are shown below.

$$\begin{array}{ll}
(\mathbf{T}\,\exists) & \text{provable}(\mathit{\Pi}\,{:}\,\mathbf{T}\,\exists yR, \theta_0, \theta_1) \text{ if} \\
& \qquad \text{provable}(\mathit{\Pi}\,{:}\,\mathbf{T}\,[!f(?x_1, \ldots, ?x_m)/y]R, \theta_0, \theta_1) \\
(\mathbf{F}\,\exists) & \text{provable}(\mathit{\Pi}\,{:}\,\mathbf{F}\,\exists xR, \theta_0, \theta_1) \text{ if} \\
& \qquad \text{provable}((?x \diamond \mathit{\Pi}\,{:}\,\mathbf{F}\,[?x/x]R) \diamond \mathit{\Pi}\,{:}\,\mathbf{F}\,\exists xR, \theta_0, \theta_1)
\end{array}$$

where: $?x$ is a new variable, and it has type ν if the quantifier has type $((\nu \rightarrow o) \rightarrow o)$; and $!f$ is a new function constant, with type $(\nu_1 \rightarrow \ldots \rightarrow \nu_m \rightarrow \nu)$ when the quantifier has type $((\nu \rightarrow o) \rightarrow o)$, $\mathit{\Pi}$ is the tuple $\langle x_1, \ldots, x_m \rangle$, and each x_i $(0 < i \leq m)$ has type ν_i. Notice that: $!f(?x_1, \ldots, ?x_m)$ is the application of $!f$ to the tuple $\langle ?x_1, \ldots, ?x_m \rangle$, i.e., $\mathit{\Pi}$,and $[\alpha/x]R$ is the result of replacing all free occurrences of x in R by α.

Finally, there are the rules: (**axiom**), which is used to close a branch in a proof tree, and the structural (**exchange**), which is necessary because tuples are used in the formalization.

$$\begin{array}{ll}
(\textbf{axiom}) & \text{provable}(\mathit{\Pi}\,{:}\,\mathbf{T}\,\alpha \diamond \mathit{\Pi}\,{:}\,\mathbf{F}\,\alpha' \diamond \mathit{\Gamma}, \theta_0, \theta_1) \quad \text{if} \\
& \qquad \text{unify}(\{\langle \alpha, \alpha' \rangle\} \cup \theta_0, \theta_1) \\
(\textbf{exchange}) & \text{provable}(\beta \diamond \mathit{\Gamma}, \theta_0, \theta_1) \text{ if} \\
& \qquad \text{provable}(\mathit{\Gamma} +\!\!+ \beta, \theta_0, \theta_1)
\end{array}$$

where β is a signed formula, and $\mathit{\Gamma} +\!\!+ \beta$ is the tuple that results when β is added at the end of $\mathit{\Gamma}$, itself a tuple.

The system defines rules for rewriting $(\alpha = \beta)$ to $\forall R(R(\alpha) \supset R(\beta))$.

Admissibility Rule: *A goal can be admitted to proof (through* **provable** *defined above) if and only if the free variables occurring in any of its formulas were introduced by quantifier elimination rules.*

An important concept used in the definition of $\mathcal{P}_2$ is the *Skolem instance* of a formula. The *Skolem instance* of a formula is either the formula itself, or any formula obtained from it by eliminating one or more of its quantifiers, using the inference rules *provided by* the formalization of *the given logic*. A *Skolem instance* of a goal $\mathit{\Gamma}$ is another goal $\mathit{\Gamma}'$ such that $\alpha' \in \mathit{\Gamma}'$ if and only if α' is the Skolem instance of some $\alpha \in \mathit{\Gamma}$.

4 The $\mathcal{P}_2$ Algorithm

Figure 3 shows the $\mathcal{P}_2$ algorithm. $\mathcal{P}_2$ receives as input a set of proof *steps* (W in Fig.3), and a set of fixed terms (F in Fig.3) of the logic, that will contain, for example, logical constants. It generates a set of *methods* (represented by M in Fig.3), defining a proof search strategy.

Proofs are decomposed in *steps*. A *step* is an ordered pair containing a goal and the name of an inference rule. A *method* is a triple, containing a *meta-goal*, a set of negative conditions (a set of signed formulas), and the name of an inference rule. A *meta-goal* is a tuple of signed formulas (without prefixes.)

```
Algorithm $\mathcal{P}_2$
Input: $W$: set of steps; $F$: set of fixed terms
Output: set of methods
let $M = \{$schematization$(x,F)$: $x \in W\}$;
set $M$ to $\{x \in M$: not redundant$(x,M,W)\}$;
do $n$ times
    let $L = M$
    for each $x$ in $M$
        for each $y$ generated from $x$
            if acceptable$(y,x,M,W)$
            then set $L$ to $L \cup \{y\}$;
    endfor;
    set $M$ to $\{x \in L$: not redundant$(x,L,W)\}$;
enddo;
return $M$.
```

Fig. 3. The $\mathcal{P}_2$ Algorithm

$\mathcal{P}_2$ can be seen as an evolutionary algorithm for generating methods from steps. In the beginning, one method is generated for each step, by a process of *schematization* (defined below). After that, a cycle is repeated where new methods are generated from old ones by randomized algorithms, and methods are evaluated with respect to a criterion of *redundancy*, redundant methods being discarded. The number of repetitions of the cycle (n in the figure) was set to 10 in the experiments reported in this paper.

4.1 Matching, Confirmation, and Contradiction

Method matching is one of the fundamental concepts of $\mathcal{P}_2$. This concept establishes the interface between $\mathcal{P}_2$ and the logic to which $\mathcal{P}_2$ is applied. *Method matching* is used to define the concepts of *confirmation* and *contradiction*, used by $\mathcal{P}_2$ to evaluate the strength of methods. In the definition below, len(Γ) is the number of formulas in Γ, drop(Γ, n) is a sub-tuple of Γ without its n initial formulas, and zip(Γ_0, Γ_1) is a set of pairs obtained by pairing element i of Γ_0 with element i of Γ_1, until exhausting the shortest of them. Γ' is a permutation of Γ when, for all α, $\alpha \in \Gamma'$ if and only if $\alpha \in \Gamma$.

Definition 1. *A* ***match*** *between a method $\langle \Gamma_m; \Delta_m; R_m \rangle$ and a goal Γ_g is a pair (Γ, θ) such that for some Γ'_g all of the following conditions hold:*

- *Γ'_g is a permutation of a Skolem instance Γ''_g of Γ_g;*
- *$len(\Gamma_m) \leq len(\Gamma'_g)$;*
- *$unify(zip(\Gamma_m, \Gamma'_g), \theta)$*
- *$\Gamma = \theta(\Gamma'_g)$;*
- *there is no θ' such that for some $\alpha_d \in \Delta_m$, $unify(\{\langle \alpha_d, \beta' \rangle\} \cup \theta, \theta')$, where β' is Skolem instance of some β occurring in*

$$drop(\Gamma, len(\Gamma_m)).$$

In the matching between method $(\mathfrak{m}_1)$ and goal $(\mathcal{G}_2)$, a Skolem instance for $(\mathcal{G}_2)$ is $(\mathcal{G}_3)$, which would be the Γ''_g of the definition above. Its permutation, Γ'_g in the definition, would be $(\mathcal{G}_4)$.

$$\langle\langle \mathbf{T}\,(\mathcal{A} \supset \mathcal{B}), \mathbf{T}\,\mathcal{A}\rangle; \{\mathbf{F}\,\mathcal{A}\}; (\mathbf{T}\ \supset)\rangle \tag{$\mathfrak{m}_1$}$$
$$\langle () : \mathbf{F}\,\exists x q(x), () : \mathbf{T}\,\exists x p(x), () : \mathbf{T}\,\forall x (p(x) \supset q(x))\rangle \tag{$\mathcal{G}_2$}$$
$$\langle () : \mathbf{F}\,\exists x q(x), () : \mathbf{T}\,p(!a), (?x_0) : \mathbf{T}\,(p(?x_0) \supset q(?x_0))\rangle \tag{$\mathcal{G}_3$}$$
$$\langle (?x_0) : \mathbf{T}\,(p(?x_0) \supset q(?x_0)), () : \mathbf{T}\,p(!a), () : \mathbf{F}\,\exists x q(x)\rangle \tag{$\mathcal{G}_4$}$$

The unifier θ is the solution of
$$\{\langle \mathbf{T}\,(\mathcal{A} \supset \mathcal{B}), \mathbf{T}\,(p(?x_0) \supset q(?x_0))\rangle, \langle \mathbf{T}\,\mathcal{A}, \mathbf{T}\,p(!a)\rangle\},$$
that gives $\{\langle ?x_0, !a\rangle, \langle \mathcal{A}, p(!a)\rangle, \langle \mathcal{B}, q(!a)\rangle\}$. Given that $\theta(\mathbf{F}\,\mathcal{A})$ (where $\mathbf{F}\,\mathcal{A}$ is the only formula in the negative conditions) does not have an occurrence in
$$\langle () : \mathbf{F}\,\exists x q(x)\rangle \qquad ,$$
the resulting match is the pair (Γ, θ), where θ is as above, and Γ is
$$\langle () : \mathbf{T}\,(p(!a) \supset q(!a)), () : \mathbf{T}\,p(!a), () : \mathbf{F}\,\exists x q(x)\rangle.$$

Definition 2. *A step $\mathfrak{s} = \langle \Gamma_s; R_s\rangle$* ***is a confirmation for*** *a method $\mathfrak{m} = \langle \Gamma_m; \Delta; R\rangle$ (denoted $\mathfrak{s} \models \mathfrak{m}$) if and only if both R is equal to R_s and there is a match between $\mathfrak{m}$ and Γ_s.*

The set of confirmations of a set of methods M in a set of steps W is
$$\mathcal{K}(M, W) = \{\mathfrak{s} \in W : \exists \mathfrak{m} \in M.\mathfrak{s} \models \mathfrak{m}\}.$$

Definition 3. *A step $\mathfrak{s} = \langle \Gamma_s; R_s\rangle$* ***is a contradiction for*** *a method $\mathfrak{m} = \langle \Gamma_m; \Delta; R\rangle$ (denoted $\mathfrak{s} \dashv \mathfrak{m}$) if and only if both R is different from R_s and there is a match between $\mathfrak{m}$ and Γ_s.*

4.2 Generating Methods

The first set of methods produced by $\mathcal{P}_2$ is a result of uniformly replacing variable terms occurring in steps by fresh meta-variables: this is called **Schematization**. The *Schematization* of the step represented by $\langle\langle \mathbf{T}\,p, \mathbf{F}\,p\rangle; (\mathbf{axiom})\rangle$ is $\langle\langle \mathbf{T}\,\mathcal{A}, \mathbf{F}\,\mathcal{A}\rangle; \{\}; (\mathbf{axiom})\rangle$.

Definition 4. *The* ***schematization of a goal*** *Γ w.r.t. to a set of fixed terms F is the goal obtained by uniformly replacing each atomic term t occurring in Γ, such that $t \notin F$ and t is not bound, by a new variable with the same type as t.*

The ***schematization of a step*** *$\langle \Gamma; R\rangle$ w.r.t. to a set of fixed terms F is the method $\langle \Gamma'; \Delta; R\rangle$, where Γ' is a schematization of Γ, and $\Delta = \emptyset$.*

New methods can be generated from other methods by generalization procedures and one strengthening procedure, which creates negative conditions. The processes of generalization widen the set of steps matched by one method, possibly producing overlapping methods, while the strengthening procedure reduces the scope of application of a method, and it should be applied to reduce its set of contradictions.

Definition 5. *A method* $\mathfrak{m} = \langle \Gamma; \Delta; R_0 \rangle$ *is a* ***negative strengthening*** *of a method* $\mathfrak{m}_0 = \langle \Gamma_0; \Delta_0; R_0 \rangle$ *with respect to a step* $\mathfrak{s} = \langle \Gamma_s; R_s \rangle$ *if and only if the following conditions all hold:*

- *there is a match* (Γ_m, θ_m) *between* $\mathfrak{m}_0$ *and* Γ_s*;*
- Γ' *is a schematization of* Γ_s*;*
- Γ *is an initial sub-tuple of* Γ' *such that* $len(\Gamma) = len(\Gamma_0)$*;*
- $\Delta = \{\alpha \in \Gamma' : \alpha \notin \Gamma\}$.

The method ($\mathfrak{m}_7$), next, could be generated by *negative strengthening* of ($\mathfrak{m}_5$) with respect to step ($\mathfrak{s}_6$).

$$\langle\langle \mathbf{T}(\neg\mathcal{A} \supset \bot), \mathbf{T}(\neg\mathcal{B} \supset \bot), \mathbf{T}\,\mathcal{B}, \mathbf{T}(\neg(\mathcal{B} \wedge \mathcal{A})), \mathbf{F}\,\bot\rangle; \{\}; (\mathbf{T} \supset)\rangle \tag{$\mathfrak{m}_5$}$$

$$\langle\langle \mathbf{T}(\neg(p \wedge q)), \mathbf{T}(\neg p \supset \bot), \mathbf{T}\,p, \mathbf{T}(\neg q \supset \bot), \mathbf{T}\,q, \mathbf{F}\,\bot\rangle; (\mathbf{T}\,\neg)\rangle \tag{$\mathfrak{s}_6$}$$

$$\langle\langle \mathbf{T}(\neg\mathcal{A}_0 \supset \bot), \mathbf{T}(\neg\mathcal{B}_0 \supset \bot), \mathbf{T}\,\mathcal{B}_0, \mathbf{T}(\neg(\mathcal{B}_0 \wedge \mathcal{A}_0)), \mathbf{F}\,\bot\rangle; \{\mathbf{T}\,\mathcal{A}_0\}; (\mathbf{T} \supset)\rangle \tag{$\mathfrak{m}_7$}$$

Definition 6. *A method* $\mathfrak{m}$ *is* ***generated*** *from a method* $\mathfrak{n}$ *if and only if* $\mathfrak{m}$ *is the result of applying to* $\mathfrak{n}$ *one of the following procedures:*

Thinning, *which consists of dropping any meta-formula occurring in either the meta-goal or the negative conditions of* $\mathfrak{n}$ *;*
Random lifting, *which replaces one occurrence of a term in* $\mathfrak{m}$ *by a new variable of the same type;*
Uniform lifting, *that replaces all occurrences of one term by a unique new variable of the same type;*
Negative strengthening *with respect to a step* $\mathfrak{s}$.

$$\langle\langle \mathbf{T}\,\mathcal{A}, \mathbf{T}\,\mathcal{A} \supset \bot, \mathbf{F}\,\mathcal{A}\rangle; \{\}; (axiom)\rangle \tag{$\mathfrak{m}_8$}$$

$$\langle\langle \mathbf{T}\,\mathcal{A}, \mathbf{F}\,\mathcal{A}\rangle; \{\}; (axiom)\rangle \tag{$\mathfrak{m}_9$}$$

$$\langle\langle \mathbf{T}\,\forall x.(\mathcal{P}(\mathcal{G}(\mathcal{A}), x) \supset \mathcal{Q}(\mathcal{B}, \mathcal{G}(\mathcal{A})))\rangle; \{\}; (\mathbf{T}\forall)\rangle \tag{$\mathfrak{m}_{10}$}$$

$$\langle\langle \mathbf{T}\,\forall x.(\mathcal{P}(\mathcal{G}(\mathcal{A}), x) \supset \mathcal{Q}(\mathcal{B}, \mathcal{G}(\mathcal{C})))\rangle; \{\}; (\mathbf{T}\forall)\rangle \tag{$\mathfrak{m}_{11}$}$$

$$\langle\langle \mathbf{T}\,\forall x.(\mathcal{P}(\mathcal{C}, x) \supset \mathcal{Q}(\mathcal{B}, \mathcal{C}))\rangle; \{\}; (\mathbf{T}\forall)\rangle \tag{$\mathfrak{m}_{12}$}$$

$$\langle\langle \mathbf{T}\,\forall x.(\mathcal{R}(x) \supset \mathcal{Q}(\mathcal{B}, \mathcal{G}(\mathcal{A})))\rangle; \{\}; (\mathbf{T}\forall)\rangle \tag{$\mathfrak{m}_{13}$}$$

For example, given the methods above, from ($\mathfrak{m}_8$), ($\mathfrak{m}_9$) could be generated by *thinning*; *random lifting* could be used to generalize ($\mathfrak{m}_{10}$) to ($\mathfrak{m}_{11}$), with $\mathcal{C}$ replacing the second occurrence of $\mathcal{A}$; *uniform lifting* could be used to generalize ($\mathfrak{m}_{10}$) to either ($\mathfrak{m}_{12}$) (by substituting $\mathcal{C}$ for $\mathcal{G}(\mathcal{A})$) or ($\mathfrak{m}_{13}$) (by substituting $\mathcal{R}$ for $(\mathcal{P}(\mathcal{G}\mathcal{A}))$.

These procedures give $\mathcal{P}_2$ the characteristic of randomized search: the formulas to be dropped or replaced are chosen at random. In the next section, a structure is defined to order methods in a default hierarchy, which is used to discard superfluous or redundant methods and to give priority to the best ones.

4.3 $\mathcal{Q}_1$: Precedence among Methods

A structure will be defined that will be used to partition a set of methods M into levels, such that contradictions for methods in one level are confirmations for methods in levels preceding it. Such partition is founded on the idea (already studied by Kleene for Intuitionistic Logic in [8] and also [14]) that for many logics there are certain inferences that should always take precedence over any other inferences. However, since many alternative criteria can be used to perform such partition, a general structure ($\mathcal{Q}_\star$), parameterized with respect to a partitioning function Φ, is introduced first.

Definition 7. *The $\mathcal{Q}_\star$ structure for a set of methods M and a set of steps W* (denoted $\mathcal{Q}_\star(\Phi, (M; W))$) *is a family of subsets of M indexed by the map*

$$\Sigma : \mathbb{N} \to 2^M,$$

where $\mathbb{N}$ is the set of non-negative integers, 2^M is the power set of M, and:

1. let $\sigma_0 = \Phi(M, W)$

$$\Sigma(0) = \begin{cases} M, & \textit{if } \sigma_0 = \emptyset \\ \sigma_0, & \textit{otherwise} \end{cases}$$

2. for any $i \geq 0$, let $\sigma_{i+1} = \Phi(M - \Sigma(i), W - \mathcal{K}(\Sigma(i), W))$

$$\Sigma(i+1) = \begin{cases} M, & \textit{if } \sigma_{i+1} = \emptyset \\ \Sigma(i) \cup \sigma_{i+1}, & \textit{otherwise} \end{cases}$$

Given an adequate criterion (through parameter Φ), this structure can produce Σ as a chain of subsets of M ordered w.r.t. set inclusion. Φ_p, defined below, separates from a set of methods M all methods that do not have contradictions in W. Assuming that rule($\mathfrak{m}$) (rule($\mathfrak{s}$)) denotes the inference rule recommended by a method $\mathfrak{m}$ (used in a step $\mathfrak{s}$), the function Φ_t separates a subset of a given set of methods M, such that each method in the subset is confirmed by all steps that used the inference rule recommended by the method.

Definition 8.

$$\Phi_p(M, W) = \{\mathfrak{m} \in M | \neg(\exists \mathfrak{s} \in W. \mathfrak{s} \mathrel{=\!\!|} \mathfrak{m})\}$$
$$\Phi_t(M, W) = \{\mathfrak{m} \in M : \forall \mathfrak{s} \in W.\ \textit{if rule}(\mathfrak{s}) = \textit{rule}(\mathfrak{m}) \textit{ then } \mathfrak{s} \models \mathfrak{m}\}$$

The structure defined by $\mathcal{Q}_\star(\Phi_p, (M; W))$ would partition the set of methods in the chain Σ, such that methods in one level (say $\Sigma(i)$) would always have contradictions confirmed by methods in levels above ($\Sigma(i-j), j > 0$). A more restrictive criterion, however, is used to define a structure called $\mathcal{Q}_1$. This criterion gives preference to more general methods, through the function Φ_t. Φ_t is composed with Φ_p, through Φ_W. The function Φ_W below denotes a set of functions: one for each W.

Definition 9.

$$\Phi_W(M) = \begin{cases} \Phi_t(M,W) & \textit{if } \Phi_t(M,W) \neq \emptyset \\ M & \textit{otherwise} \end{cases}$$

$$\mathcal{Q}_1(M;W) = \mathcal{Q}_\star(\Phi_W \circ \Phi_p, (M;W)),$$

where $\Phi_W \circ \Phi_p$ denotes the composition of Φ_W with Φ_p.

Definition 10. *The* ***level*** *of a method $\mathfrak{m}$ in a structure $\mathcal{Q}_1(M;W)$* (denoted level($\mathfrak{m}, \mathcal{Q}_1(M;W)$)) *is the least i such that $\mathfrak{m} \in \Sigma(i)$.*

Definition 11. *A method $\mathfrak{m}$ is* ***redundant*** *in a structure $\mathcal{Q}_1(M;W)$ if and only if there is another method $\mathfrak{m}' \in m$ such that both*

- *level($\mathfrak{m}', \mathcal{Q}_1(M;W)$) $\leq$ level($\mathfrak{m}, \mathcal{Q}_1(M;W)$), and*
- *$\mathcal{K}(\{\mathfrak{m}\}, W) \subseteq \mathcal{K}(\{\mathfrak{m}'\}, W)$*

Definition 12. *A method $\mathfrak{m}$ is* ***acceptable*** *in comparison with $\mathfrak{n}$ with respect to $\mathcal{Q}_1(M;W)$ if and only if for some $k \geq 0$*

$$\textit{level}(\mathfrak{m}, \mathcal{Q}_1(M;W)) \leq \textit{level}(\mathfrak{n}, \mathcal{Q}_1(M;W)) + k.$$

The parameter k must be determined by experiment. It must be noticed, however, that acceptability is incorporated in $\mathcal{P}_2$ mainly for the sake of efficiency: methods which would be deemed unacceptable tend to have a high number of contradictions and to be located in the deepest level of $\mathcal{Q}_1$, being eliminated by the redundancy criterion.

5 Experiments with $\mathcal{P}_2$

$\mathcal{P}_2$ has been implemented on the top of a platform, the *Pframe*, which is composed of modules implementing: the syntax of the $\lambda \rightarrow$; Huet's higher-order unification algorithm ([7]) with Miller's occurrence check ([12]); the concepts of goals and proof steps; method matching that can generate a lazy list of matches between a method (or a list of methods) and a goal; and the concept of proof tree, with procedures for depth-first, iterative deepening and breadth search. A prover is constructed for a logic $\mathcal{L}$ by linking *Pframe* with a module implementing the inference rules for $\mathcal{L}$, and a set of methods generated by $\mathcal{P}_2$.

The search tree for a given proof is defined by the selected search procedure (depth-first, iterative deepening, or breadth first), and the set of methods. In depth-first search, a prover behaves very much like a standard Prolog. $\mathcal{P}_2$ orders the methods by priority (level 0 of $\mathcal{Q}_1$ first.) The prover uses the methods in the given order, and maintains a list of goals that is expanded in a left-to-right order (backtracking being, of course, part of this search procedure.)

At the kernel of the *driver* of $\mathcal{P}_2$-hol, there is a procedure that generates recommendations of inference rules to be applied to given goals. This procedure can also be seen as implementing a relation, like:

$$\text{recommend}((\Gamma_g : \theta_g), \mathfrak{m}, \langle(\Gamma : \theta), R\rangle),$$

that recommends the application of R to goal Γ, where:

– Γ_g is the goal to be solved under restriction θ_g.
– $\mathfrak{m}$ is a method $\langle \Gamma_m; \Delta_m; R\rangle$.
– $(\Gamma : \theta)$ is a match between $\mathfrak{m}$ and $(\Gamma_g : \theta_g)$.
– R is the recommended rule.

Proofs can be performed in either interactive, or fully automatic mode.

1. $\vdash (p \supset (q \supset p))$
2. $\vdash ((p \wedge (p \supset q)) \supset q)$
3. $p, (p \supset q), (q \supset r) \vdash r$
4. $\vdash (\neg(p \wedge q) \supset (\neg p \vee \neg q))$
5. $\vdash (p \vee \neg p)$
6. $(\neg p \wedge p) \vdash (q \vee r)$
7. $\vdash ((p \wedge q) \supset (p \vee r))$
8. $(p \supset q), \neg q, (p \wedge r) \vdash$
9. $\vdash ((p = \neg p) \supset q), \exists X Q_1(X)$
10. $(p \wedge q), (r \supset \neg r), ((A = A) \supset (B = A)) \vdash p, \exists X Q_1(X)$
11. $(p \wedge q), ((r \vee s) \supset (Q_1(A) \wedge S_1(B))), r \vdash q$
12. $(p \supset q), (p \vee p), (r \supset r) \vdash q$
13. $((p \vee r) \supset q), p, (s \supset r), (r \supset r) \vdash q$
14. $(r \wedge s), (q \supset p) \vdash \neg(\neg p \wedge p)$
15. $(p \vee q) \vdash ((q \vee r) \vee p)$
16. $\vdash \forall X((P_1(X) \wedge (P_1(X) \supset \neg Q_1(X))) \supset \neg Q_1(X))$
17. $\vdash (p \supset (A = A))$
18. $(A = B) \vdash (B = A)$
19. $p, (A = B) \vdash (G_1(A) = G_1(B))$
20. $((p \vee r) \supset (Q_1(A) \wedge s)), p \vdash \exists X Q_1(X)$
21. $\vdash \neg\exists X Q_1(X), \exists X Q_1(X)$

Fig. 4. A training set for $\mathcal{P}_2$

5.1 An Experiment with Second-Order Logic

An experiment was conducted with $\mathcal{P}_2$ to generate a prover for second-order logic. The training set was the one given in Fig.4. A set of problems, proved using the generated set of methods, is in Fig.1. The two sets were chosen to assess $\mathcal{P}_2$'s ability to generate heuristic provers that can deal with potentially explosive proofs. An important characteristic of the training set is the simplicity of the theorems contained in there, most of them, theorems of the propositional calculus. This is a very important feature of the set because it provides evidence in support of the idea that potentially explosive proofs (as proofs with higher-order variables usually are) can be controlled, using proof search strategies generated from propositional or first-order examples. It is important to observe also that the examples were chosen to teach $\mathcal{P}_2$ the proof skills that are taught in any introductory text of logic, essentially including: splitting conjunctions, as in problem 7; identifying contradictions, as in problem 6; using the sequent calculus (or tableau) equivalents of modus ponens and modus tolens, as in problem 3; and the elimination of quantifiers. The timings and numbers of inferences used by the prover (with a $\mathcal{P}_2$ generated proof search strategy) are in Tab.(1). The experiment was conducted with an implementation of $\mathcal{P}_2$-hol written in Clisp, and running under Linux on a PC 486, 66MHz. Several of those problems can be considered as hard. The author is not aware of any prover that, using a machine generated proof search strategy, has been able to prove in automatic mode, for example, the induction principles of problems 13 and 14 (taken from [3]), or the Cantor's theorem of problem 15. Cantor's theorem has been proved before in automatic mode by TPS, using a human-coded plan ([1].)

Thm	Infs	Secs	Thm	Infs	Secs
1	27	8.61	9	9	3.50
2	19	4.97	10	4	0.28
3	6	0.95	11	12	4.37
4	5	1.20	12	2	3.67
5	2	0.25	13	5	2.14
6	5	0.90	14	54	34.93
7	17	3.18	15	12	8.39
8	5	0.97			

Table 1. $\mathcal{P}_2$-prover applied to set in figure 1

5.2 Important Features of $\mathcal{P}_2$ Strategies

A few examples, from Fig. 1, are analyzed to highlight the importance of some features of the strategies generated by $\mathcal{P}_2$.

Proof by Analogy. A $\mathcal{P}_2$ strategy can accommodate proofs by analogy. For example, problem 3 of Fig.1 was proved by $\mathcal{P}_2$-hol using exactly the same sequence of inferences found in the training set (Fig.4) for problem 19.

The Importance of Priority. The proof of problem 5 (Fig.1) illustrates the importance of the assignment of priorities to the generated set of methods. It started with an application of ($\mathbf{F}\ \supset$) (possible thanks to the elimination of the two external quantifiers during *method matching*), producing:

$$\langle (?A_0) : \mathbf{T}\, ?A_0(!f(?A_0)), (?A_0) : \mathbf{F}\,(!f(?A_0) \leq 0)\rangle$$

$?A_0$ is a free variable and it can be instantiated to any term (including a term with new quantifiers.) However, the method with highest priority generated by $\mathcal{P}_2$ was ($\mathfrak{m}_{14}$), that had no contradictions in the training set. A match between

($\mathfrak{m}_{14}$) and the goal above lead to a solution in one step (through the recommended inference rule (**axiom**).)

$$\langle\langle \mathbf{T}\,\mathcal{A}, \mathbf{F}\,\mathcal{A}\rangle; \{\}; (\mathbf{axiom})\rangle \qquad (\mathfrak{m}_{14})$$

Skolem Instance. The importance of the concept of *Skolem instance* can be illustrated by the proof of the problem 12, that again started with an application of ($\mathbf{F} \supset$) to produce

$$\langle ()\colon \mathbf{T}\,\forall X(\exists U(X = (2 * U)) \leftrightarrow \neg\exists V((!a + 1) = (2 * V))),$$
$$()\colon \mathbf{F}\,\exists A \forall X(A(X) \leftrightarrow \neg A(X + 1))\rangle.$$

In the next step, the method with highest priority was again $\mathfrak{m}_{14}$, but a match was only possible after the elimination of the external quantifiers present in the two formulas of the goal above, which produced:

$$\langle (?x_0)\colon \mathbf{T}\,\exists U(?x_0 = (2 * U)) \leftrightarrow \neg\exists V((?x_0 + 1) = (2 * V)),$$
$$(?A_0)\colon \mathbf{F}\,(?A_0(!f(?A_0)) \leftrightarrow \neg ?A_0(!f(?A_0) + 1))\rangle.$$

The proof above took only 2 steps with $\mathcal{P}_2$-hol. It was presented with 14 steps in [3]. Problem 12, after skolemization, and conversion to clausal notation, produced 4 clauses in [3]. This contrasts with the fact that in the proof obtained with $\mathcal{P}_2$-hol, at most 2 formulas are present at any time. The approach taken with $\mathcal{P}_2$-hol, which consists in a *lazy* elimination of quantifiers (quantifiers are removed only when they appear as external) seems to contribute to a reduction in the number of formulas introduced. In addition, the quantifier elimination followed by unification with methods acquired from previous experience helps in many cases to find terms to instantiate free variables as soon as they are introduced.

These factors played an important role in the proofs for problems 13 and 14. Problem 14 was proved in 54 steps. This is apparently worse than the 25 presented in [3]. However, notice that $\mathcal{P}_2$-hol has no special axioms for dealing with sets, and that [3] presents just the final proof obtained, without commenting on *redundant* inferences performed by the prover. The 54 steps reported for $\mathcal{P}_2$-hol included steps that were discarded by backtracking.

6 Properties of the Generated Provers

The most important property of the provers generated with $\mathcal{P}_2$ is the fact that their *soundness* depends exclusively on the formalization of the logic in question. This is achieved in $\mathcal{P}_2$ through the concept of *method match*, which contains the interface between $\mathcal{P}_2$ and the given logic. This interface is more exactly located in the definitions of *Skolem instance*, and *unification*, both assumed as being provided by the logic. $\mathcal{P}_2$ has been used, for example, with propositional modal logics S4 and S5, where the *Skolem instance* of a formula is the formula itself, and the unification is indeed a first-order matching procedure. The result of a *method match* is a pair, containing a permutation of the input goal and a substitution. Soundness is easily guaranteed in this case.

In the case of $\mathcal{P}_2$-hol, the soundness is guaranteed in the formalization of the logic through the *admissibility rule*, which basically imposes that variables occurring in any method are kept separated from variables occurring in goals: methods' variables will not affect the dependencies that should exist between parameters introduced by quantifier rules and free variables occurring in the prefixes of any goal. In [10], the soundness of $\mathcal{P}_2$-hol itself is discussed. $\mathcal{P}_2$-hol is also compared in [10] with Miller's $\mathcal{T}$ ([11]), a sound formalization of a sub-logic of Church's type theory that excludes the axioms of choice, extensionality, descriptions, and infinity. Differently, $\mathcal{P}_2$-hol is a sound formalization of a sub-logic of Church's type theory that includes the axiom of choice. (Hintikka's defense, in [6], of formalizations of second-order logic that assume the axiom of choice is discussed in [10].)

Another important property of the methodology underlying $\mathcal{P}_2$ is in its *generality*. $\mathcal{P}_2$ does not impose many restrictions on the formalizations of the target logics. It has been used to generate provers for **IPC**, modal logics S4, S5, and second-order logic. This provides, probably, a wider range of applications than that found for any other study of Machine Learning applied to automated theorem proving.

Re-usability is a distinctive feature of the provers generated with $\mathcal{P}_2$. A proof search strategy generated by $\mathcal{P}_2$ combines the knowledge of several proofs, and, as such can be applied to a wide range of new problems. In addition, $\mathcal{P}_2$ provers do not need special procedures to implement patches for cases when a proof plan fails: each structure $\mathcal{Q}_1$ can accomodate several methods with different levels of generality that recommend the same inference rule.

7 Conclusions and Related Work

$\mathcal{P}_2$ is an algorithm of the family $\mathcal{P}_\star$. $\mathcal{P}_\star$ was introduced first in [9]. The first algorithm presented in there was $\mathcal{P}_0$. $\mathcal{P}_0$ was heavily based on Tarver's **M2**([16]), which has itself important similarities with the Version Spaces methodology ([13]). **M2** used limited forms of *thinning* and *random lifting* to generate *tactics*, *tactic* being the name given by Tarver to a method without negative conditions, and whose meta-goal could contain only one **F**-signed formula. **M2** defined *fitness* of a *tactic* as the difference between its numbers of confirmations and contradictions, and it used *fitness* to discard *unacceptable tactics*. $\mathcal{P}_0$ and **M2** are discussed in [10], where they are compared with $\mathcal{P}_{1+}$ (another $\mathcal{P}_\star$ algorithm) and $\mathcal{P}_2$. A priority queue for classifying methods was first described in [9].

The $\mathcal{P}_\star$ algorithms comprise an effective methodology for applying machine learning to automated theorem proving. $\mathcal{P}_2$ has been probably applied to a wider range of logics than any other methodology combining ML with theorem proving. In addition, they seem to offer a clear advantage over the case-based reasoning approach of analogical proving systems, which in general demands, first, the interference of a user in the selection of one source problem for each target analogue that must be proved, and, second, complicated procedures for patching flawed plans. $\mathcal{P}_2$ can produce generic provers for a target logic, as it has been done for **IPC**, and modal logics S4 and S5 (see experiments in [10]), and it can produce specialist provers for one specific domain (set theory, for example, in the experiments of this paper.)

It is shown in [10] that Angluin's *training sequences* (a criterion for the learnability of programs from graphs of functions introduced in [2]) are a special case of the structures generated by $\mathcal{Q}_1$. [10] also introduces $\mathcal{Q}_2$, that further generalizes $\mathcal{Q}_1$. It remains to be shown whether a program exists, that can be learned with $\mathcal{Q}_2$, and it cannot be learned with $\mathcal{Q}_1$.

The structures generated by $\mathcal{Q}_1$ can be compiled into a Petri net, which could be used to produce parallel provers. This compilation is discussed also in [10]. [10] also discusses how $\mathcal{P}_2$ can be applied to resolution-based provers.

Acknowledgements

I would like to thank Dr. Pat Hill (University of Leeds), and Dr. Alan Smaill (University of Edinburgh) for many and important comments that greatly helped in the presentation of my thesis and this paper.

References

1. Peter B. Andrews, Dale A. Miller, Eve L. Cohen, and Frank Pfenning, *Automating higher-order logic*, In Bledsoe and Loveland [4], pp. 169–192.
2. Dana Angluin, William Gasarch, and Carl H. Smith, *Training sequences*, Theoretical Computer Science **66** (1989), 255–272.
3. W.W. Bledsoe and Guohui Feng, *Set-var*, Journal of Automated Reasoning **11** (1993), 293–314.
4. W.W. Bledsoe and D.W. Loveland (eds.), *Automated theorem proving: After 25 years*, American Mathematical Society, Providence – Rhode Island, 1984.
5. Alonzo Church, *A formulation of the simple theory of types*, Journal of Symbolic Logic **5** (1940), 56–68.
6. Jaakko Hintikka, *Truth definition, Skolem functions, and axiomatic set theory*, The Bulletin of Symbolic Logic **4** (1998), no. 3, 303–337.
7. Gérard P. Huet, *A unification algorithm for typed λ-calculus*, Theoretical Computer Science **1** (1975), 27–57.
8. Stephen Cole Kleene, *Permutability of inferences in Gentzen's calculi LK and LJ*, Memoirs of the AMS, vol. 10, American Mathematical Society, 1952.
9. Raul H.C. Lopes, *Inducing search methods from proofs*, Tech. report, School of Computer Studies, University of Leeds, 1997.
10. Raul Henriques Cardoso Lopes, *Inductive generalization of proof search strategies from examples*, Ph.D. thesis, University of Leeds, 1998.
11. Dale A. Miller, *Compact representation of proofs*, Studia Logica **4** (1987), 347–370.
12. ______, *Unification under a mixed prefix*, Journal of Symbolic Computation **14** (1992), 321–358.
13. Tom M. Mitchell, *Generalization as search*, Artificial Intelligence **18** (1982), 203–226.
14. N. Shankar, *Proof search in the intuitionistic sequent calculus*, In Stickel [15], LNAI, 449, pp. 522–536.
15. M.E. Stickel (ed.), *Proceedings of the 10th International Conference on Automated Deduction*, Springer-Verlag, 1990, LNAI, 449.
16. Mark Tarver, *An algorithm for inducing tactics from sequentzen proofs*, Workshop on practical applications of automated reasoning, 1995, AISB.

Author Index