Martin Giese
Arild Waaler (Eds.)

LNAI 5607

Automated Reasoning with Analytic Tableaux and Related Methods

18th International Conference, TABLEAUX 2009
Oslo, Norway, July 2009
Proceedings

Springer

Lecture Notes in Artificial Intelligence 5607

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Martin Giese   Arild Waaler (Eds.)

Automated Reasoning with Analytic Tableaux and Related Methods

18th International Conference, TABLEAUX 2009
Oslo, Norway, July 6-10, 2009
Proceedings

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Martin Giese
Arild Waaler
University of Oslo
Department of Informatics
P.O. Box 1080 Blindern, 0316 Oslo, Norway
E-mail: {martingi, arild}@ifi.uio.no

Library of Congress Control Number: 2009929055

CR Subject Classification (1998): I.2.3, F.4.1, I.2, D.1.6, D.2.4

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-642-02715-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-02715-4 Springer Berlin Heidelberg New York

springer.com

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12702633 06/3180 5 4 3 2 1 0

Preface

This volume contains the research papers presented at the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2009) held July 6-10, 2009 in Oslo, Norway. This conference was the 18th in a series of international meetings since 1992 (listed on page IX). It was collocated with FTP 2009, the Workshop on First-Order Theorem Proving.

The Program Committee of TABLEAUX 2009 received 44 submissions from 24 countries. Each paper was reviewed by at least three referees, after which the reviews were sent to the authors for comment in a rebuttal phase. After a final intensive discussion on the borderline papers during the online meeting of the Program Committee, 21 research papers and 1 system description were accepted based on originality, technical soundness, presentation, and relevance. Additionally, three position papers were accepted, which are published as a technical report of the University of Oslo. We wish to sincerely thank all the authors who submitted their work for consideration. And we would like to thank the Program Committee members and other referees for their great effort and professional work in the review and selection process. Their names are listed on the following pages.

In addition to the contributed papers, the program included three excellent keynote talks. We are grateful to Patrick Blackburn (INRIA Nancy, France), Peter Jeavons (Oxford University Computing Laboratory, UK), and Pierre Wolper (Université de Liege, Belgium) for accepting the invitation to address the conference. Two very good tutorials were part of TABLEAUX 2009: “The Theory of Canonical Systems” (A. Avron and A. Zamansky, Tel-Aviv University, Israel), and “LoTREC: Theory and Practice. Proving by Tableau Becomes Easier...” (Bilal Said and Olivier Gasquet, Université Paul Sabatier, Toulouse, France). We would like to express our thanks to the tutorial presenters for their contribution.

In addition to FTP 2009, three workshops were held in conjunction with TABLEAUX 2009:

- The workshop on “Tableaux Versus Automata as Logical Decision Methods,”organized by Valentin Goranko from the University of the Witwatersrand, Johannesburg, South Africa
- The workshop on “Proofs and Refutations in Non-classical Logics,” organized by Roy Dyckhoff from the University of St Andrews, Scotland, UK and Didier Galmiche from LORIA, Henri Poincaré University, Nancy, France
- The workshop “Gentzen Systems and Beyond,” organized by Kai Brünnler from the University of Bern, Switzerland, and George Metcalfe from Vanderbilt University, Nashville, USA

Abstracts of the workshop papers were published separately as a technical report of the University of Oslo.

It was a team effort that made the conference so successful. We are truly grateful to the Steering Committee members for their support. And we particularly thank the local organizers for their hard work and help in making the conference a success: Terje Aaberge, Roger Antonsen, Roar Fjellheim, Christian M. Hansen, Bjarne Holen, Magdalena Ivanovska, Espen H. Lian, Martin G. Skjæveland, Audun Stolpe, and Evgenij Thorstensen.

July 2009

Martin Giese
Arild Waaler

Organization

Program and Conference Chairs

Martin Giese	University of Oslo, Norway
Arild Waaler	University of Oslo, Norway

Program Committee

Peter Baumgartner	NICTA, Canberra, Australia
Bernhard Beckert	University of Koblenz-Landau, Germany
Christoph Benzmüller	Saarland University, Saarbrücken, Germany
Marc Bezem	University of Bergen, Norway
Torben Braüner	Roskilde University, Denmark
Agata Ciabattoni	TU Wien, Austria
Marta Cialdea Mayer	University of Rome 3, Italy
Stéphane Demri	CNRS, Cachan, France
Roy Dyckhoff	University of St Andrews, Scotland, UK
Ulrich Furbach	University of Koblenz-Landau, Germany
Didier Galmiche	LORIA, Henri Poincaré University, Nancy, France
Valentin Goranko	University of the Witwatersrand, Johannesburg, South Africa
Rajeev Goré	Australian National University, Canberra, Australia
Reiner Hähnle	Chalmers University, Göteborg, Sweden
Ullrich Hustadt	University of Liverpool, UK
Christoph Kreitz	University of Potsdam, Germany
George Metcalfe	Vanderbilt University, Nashville, USA
Neil V. Murray	University at Albany - SUNY, USA
Nicola Olivetti	Paul Cézanne University, Marseille, France
Jens Otten	University of Potsdam, Germany
Nicolas Peltier	LIG, Grenoble, France
Ulrike Sattler	University of Manchester, UK
Viorica Sofronie-Stokkermans	MPI, Saarbrücken, Germany
Frank Wolter	University of Liverpool, UK

Additional Referees

W. Ahrendt
F. Baader
T. Bolander
D. Bresolin
J. Brotherston
K. Brünnler
R. Bubel
E. Rodriguez Carbonell
S. Cerrito
M. Dam
C. Fermüller
O. Gasquet
M. Gebser
G. De Giacomo
L. Giordano
A. Haas
J.U. Hansen
M. Horridge
D. Hovland
S. Jacobs
E. Broch Johnsen
B. Konev
H. Kurokawa
M. Lange
D. Larchey-Wendling
B. Motik
D. Méry
C. Obermaier
A. Orlandini
B. Pelzer
F. Poggiolesi
A. Polonsky
G.L. Pozzato
V. Risch
G. Sandu
A. Sangnier
K. Sano
L. Santocanale
T. Schneider
L. Strassburger
T. Studer
L. Tranchini
D. Tsarkov
D. Walther
C. Weidenbach
C.-P. Wirth
A. Zamansky

Steering Committee

Rajeev Goré (President)	Australian National University, Canberra, Australia
Bernhard Beckert (Vice President)	University of Koblenz, Germany
Peter Baumgartner	NICTA, Canberra, Australia
Marta Cialdea Mayer	Universita degli studi Roma Tre, Italy
Chris Fermüller	TU Vienna, Austria
Stéphane Demri	Ecole Normale Supérieure de Cachan, France
Nicola Olivetti	Université Paul Cézanne, Marseille

Sponsoring Institutions

The Norwegian Research Council
OLF, the Norwegian Oil Industry Association
Gaz de France SUEZ
DNV
Computas AS
The Dept. of Informatics at the University of Oslo
The City of Oslo

Previous Meetings

1992	Lautenbach, Germany
1993	Marseille, France
1994	Abingdon, UK
1995	St. Goar, Germany
1996	Terrasini, Italy
1997	Pont-à-Mousson, France
1998	Oisterwijk, The Netherlands
1999	Saratoga Springs, USA
2000	St. Andrews, UK
2001	Siena, Italy (part of IJCAR)
2002	Copenhagen, Denmark
2003	Rome, Italy
2004	Cork, Ireland (part of IJCAR)
2005	Koblenz, Germany
2006	Seattle, USA (part of IJCAR)
2007	Aix-en-Provence, France
2008	Sydney, Australia (part of IJCAR)

Table of Contents

Presenting Constraints

Peter Jeavons[*]

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, UK
peter.jeavons@comlab.ox.ac.uk

Abstract. We describe the constraint satisfaction problem and show that it unifies a very wide variety of computational problems. We discuss the techniques that have been used to analyse the complexity of different forms of constraint satisfaction problem, focusing on the algebraic approach, and highlight some of the recent results in this area.

1 A Menagerie of Problems

Computational problems from many different areas involve finding values for variables that satisfy certain restrictions. In this paper we will attempt to show that it is useful to abstract the general form of such problems to obtain a generic framework known as the constraint satisfaction problem. Bringing the problems into a common framework draws attention to common aspects that they all share, and allows very general analytical approaches to be developed. We will survey some of these approaches, and the results that have been obtained in the later sections.

First, we collect a range of examples to begin to illustrate the breadth of problems that can be included under the general description of “finding values for variables that satisfy certain restrictions”. This general description leaves open the question of whether the values are chosen from a set that is finite or infinite, continuous or discrete, structured or unstructured. It also leaves open the question of how the restrictions are specified.

Example 1.1 (Linear Equations). One very important class of restrictions are those obtained by specifying a system of linear equations that must hold between certain subsets of the variables. To solve a system of simultaneous linear equations we must find values for variables that satisfy a conjunction of restrictions of this type, such as:

$$\begin{aligned} 2x_1 - 8x_3 + 5x_7 &= 3 \quad \wedge \\ x_3 - 2x_4 &= 0 \quad \wedge \\ x_2 + 2x_5 + 2x_6 + 3x_7 &= -2 \end{aligned}$$

The notion of a linear equation implicitly requires the set of values taken by the variables to have some algebraic structure: it must be possible to add and subtract multiples of these values in some way. The complexity of solving a system of such equations naturally depends on the nature of this algebraic structure.

[*] This research was supported by EPSRC research grant EP/D032636.

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 1–15, 2009.

If the set of allowable values is a *field*, allowing addition, subtraction, multiplication and division, then the problem is solvable in polynomial time, by an algorithm such as Gaussian elimination that manipulates the set of equations into an equivalent but simpler set. The field in question may be infinite (such as the rationals, $\mathbb{Q}$) or finite (such as the Galois field of order p, $\mathrm{GF}(p)$).

However, if the set of values is not a field, but a more restricted algebraic structure, such as a *ring* or *group* then we cannot use Gaussian elimination, or any algorithm that relies on using division, so the complexity is less obvious. For the ring of integers, $\mathbb{Z}$, or any other principal ideal domain, we can appeal to Bézout's identity to find a solution efficiently, but for non-negative integers, $\mathbb{N}$, the problem is easily shown to be be NP-complete, even if we allow at most 3 variables per equation [5].

For *finite* sets of possible values a more general version of this problem has recently been studied using the algebraic approach to constraint satisfaction in [32], and a complete complexity classification has been obtained.

Example 1.2 (Polynomial Equations). A more general class of problems can be obtained by relaxing the requirement for the specified restrictions to be linear, and instead allowing arbitrary polynomial equations, such as:

$$\begin{aligned} 2x_1^2 - 8x_3x_4 + 5x_7 - 3 &= 0 \quad \wedge \\ x_3 - 2x_4x_5x_7 &= 0 \quad \wedge \\ x_2^3x_3x_5^2 + 2x_5^3 + 2x_6^2 + 3x_7 + 2 &= 0 \end{aligned}$$

The notion of a polynomial again implies that the set of values taken by the variables has some algebraic structure: it must be possible to add, subtract and multiply these values in some way. The set of all polynomials over a field or ring R with variables $x_1, \ldots, x_n$, together with the standard addition and multiplication operations on polynomials, itself forms a ring, which is denoted $R[x_1, \ldots, x_n]$.

The complexity of solving a system of such equations naturally depends on the structure of this ring of polynomials. For example, when $R = \mathbb{Z}$ we are seeking integer solutions to a polynomial with integer coefficients, which is well-known to be undecidable [34]. On the other hand, when $R = \mathbb{C}$ we can decide whether a system of polynomials has any solutions by calculating a Gröbner Basis for the system, which can be done in doubly-exponential time [1]. Moreover, if the values for each individual variable are restricted to a finite set, by adding a univariate polynomial in each individual variable to the system, then a Gröbner Basis can be computed in singly-exponential time [30]. This approach is explored in more detail in [30] in connection with solution techniques for general constraint satisfaction problems over finite sets of values.

Example 1.3 (Linear Inequalities). Another general class of problems can be obtained by moving from equations to inequalities, such as:

$$\begin{aligned} 2x_1 - 8x_3 + 5x_7 &\leq 3 \quad \wedge \\ x_3 - 2x_4 &\geq 0 \quad \wedge \\ x_2 + 2x_5 + 2x_6 + 3x_7 &\leq -2 \end{aligned}$$

In this case the set of possible values must be an *ordered* set with sufficient algebraic structure to be able to add multiples of values.

If the set of possible values is $\mathbb{Q}$, then solving such a system is a sub-problem of the Linear Programming problem: it is the problem of deciding if a linear program is feasible. Since linear programming can be solved in polynomial time, this problem is clearly polynomial. However, if the set of possible values is $\mathbb{Z}$, then this problem is equivalent to Integer Programming, and hence NP-complete (unless the total number of variables is bounded, see [33]).

If the set of possible values is *finite*, then the general problem is NP-complete, but some tractable special cases have been identified using the algebraic approach to constraint satisfaction. For example, if each inequality is $\geq$ and has at most one negative coefficient, then the system can be solved in polynomial time [29].

Example 1.4 (Disequalities). Another general class of problems can be obtained by moving from inequalities to disequalities, such as:

$$\begin{aligned} 2x_1 - 8x_3 + 5x_7 &\neq 3 \quad \wedge \\ x_3 - x_4 &\neq 0 \quad \wedge \\ x_2 + 2x_5 + 2x_6 + 3x_7 &\neq -2 \end{aligned}$$

In this case, if the set of possible values is *infinite*, then a solution always exists, so the problem is trivial. However, if the set of possible values is *finite*, with at least 3 elements, then the general problem is NP-complete because we can use a system of disequalities of the form $x_i - x_j \neq 0$ to model an arbitrary Graph Colouring problem [18].

2 The Constraint Satisfaction Problem - 3 Definitions

The problems discussed in the previous section arise in many different contexts, and have been studied using a wide variety of ad hoc techniques, but they all have the same logical form. If we fix a relational structure $\mathbf{B} = (D, R_1, R_2, \ldots)$, where the universe D is the set of possible values taken by our variables, and $R_1, R_2, \ldots$ are relations over D, then each problem instance is given by a *primitive positive* sentence, Φ, involving relation symbols for the relations in $\mathbf{B}$, and the question is simply whether Φ is true in $\mathbf{B}$. A first-order formula is called primitive positive over $\mathbf{B}$ if it is of the form

$$\exists x_1 \exists x_2 \ldots \exists x_n \; \psi_1 \wedge \cdots \wedge \psi_m$$

where the ψ_i are atomic formulas, i.e., formulas of the form $R(x_{i_1}, \ldots, x_{i_k})$ where R is a relation symbol for a k-ary relation from $\mathbf{B}$.

Problems of this form are known as **constraint satisfaction problems**, and they have been very extensively studied, especially during the past 10 years (see, for example, [2,8,10,11,13,14,16,22,31]).

Definition 2.1. *An instance of the constraint satisfaction problem (CSP) is given by a primitive positive sentence, Φ, over a fixed relational structure,* $\mathbf{B}$*. The question is whether Φ is true in* $\mathbf{B}$*.*

This logical formulation of constraint satisfaction allows some classical combinatorial problems to be formulated very naturally.

Example 2.2 (Satisfiability). The standard propositional satisfiability problem for ternary clauses, 3-Sat, consists in determining whether it is possible to satisfy a Boolean formula given in CNF as a conjunction of ternary clauses.

This can be viewed as a constraint satisfaction problem by fixing the structure $\mathbf{B}_{3SAT}$ to be $(\{0,1\}, R_1, \ldots, R_8)$, where the R_i are the 8 relations definable by a single ternary clause. For example, the clause $x \vee \neg y \vee \neg z$ can be written as $R_1(x,y,z)$, where $R_1 = \{(0,0,0),(0,0,1),(0,1,0),(1,0,0),(1,0,1),(1,1,0),(1,1,1)\}$. An instance of 3-Sat corresponds to a primitive positive sentence over $\mathbf{B}_{3SAT}$ with a conjunct for each clause.

Example 2.3 (Graph Colouring). The standard Graph k-Colourability problem consists in determining whether it is possible to assign k colours to the vertices of a given graph so that adjacent vertices are assigned different colours.

This can be viewed as a constraint satisfaction problem by fixing the structure $\mathbf{B}_{kCOL}$ to be $(\{1,\ldots,k\}, \neq)$, where $\neq$ is the binary disequality relation on $\{1,\ldots,k\}$ given by $\{(i,j) : i,j \in \{1,\ldots,k\}, i \neq j\}$.

An instance of Graph k-Colouring corresponds to a primitive positive sentence over $\mathbf{B}_{kCOL}$ with a conjunct for each edge of the graph.

A related strand of research has been the development of programming languages to facilitate the expression of practical constraint satisfaction problems, and software tools to solve such problems. This approach is known as *constraint programming* [37].

In the field of constraint programming, a constraint satisfaction problem is usually defined in a more operational way, as follows.

Definition 2.4. *An instance of the constraint satisfaction problem (CSP) is given by a triple (V, Δ, C), where V is a finite set of **variables**, Δ is a function which maps each element of V to a set of possible values, called its **domain**, and C is a finite set of **constraints**.*

*Each constraint $c \in C$ is a pair, (σ, R), where $\sigma = \langle \sigma[1], \sigma[2], \ldots, \sigma[r] \rangle$ is a sequence of variables from V, called the **scope**, and $R \subseteq \Delta(\sigma[1]) \times \cdots \times \Delta(\sigma[r])$ is a **relation** that defines the assignments to the variables in σ allowed by this constraint.*

*A **solution** to a CSP instance is a function which maps each variable to a value from its domain which is consistent with all of the constraints.*

This formulation focuses attention on the *variables*, the *domains* and the *constraints*; these are the key data structures in a software system for solving constraint satisfaction problems. In this formulation the constraints are often represented by oracles - black-box algorithms for a particular constraint type, known as "propagators" [20,37], which communicate information with each other during the search for a solution by modifying the domains of the variables.

Many real-world problems such as timetabling and scheduling, are captured very naturally in this formulation, as well as classic combinatorial search problems and puzzles.

Example 2.5 (Sudoku). In a Sudoku puzzle the aim is to fill in a 9×9 grid of squares with the digits $1, \ldots, 9$ in such a way that each digit occurs exactly once in each row, each column, and each of 9 specified 3×3 subgrids. Each specific instance of the puzzle has a selection of grid entries already filled-in, and the aim is to fill in the remaining entries.

One way to model this problem as a constraint satisfaction problem (V, Δ, C) is to choose the set of variables V to be the 81 grid squares, and the function Δ to give the pre-selected value on all filled grid squares, and the set $\{1, 2, \ldots, 9\}$ on all other grid squares. The set C then contains 27 constraints, whose scopes are the 9 rows, the 9 columns, and the 9 specified 3×3 sub-grids, and whose relations are "all-different".

By using specialised propagation algorithms for the "all-different" constraints, combined with standard backtrack search, standard constraint solving software tools are able to solve (generalised) Sudoku instances extremely efficiently [40].

It is easy to translate from our second formulation of the constraint satisfaction problem (Definition 2.4) to our original formulation (Definition 2.1). To do this, simply collect together all the possible values occurring in the domains given by Δ into a single set D, and then collect together the relations over D that occur in the constraints of C, to give a relational structure $\mathbf{B}$ with universe D. The instance can then be written as a primitive positive sentence over $\mathbf{B}$ with a conjunct $R(\sigma[1], \ldots, \sigma[r])$ for each constraint (σ, R) of arity r.

In a given CSP instance there may be several constraints with the same constraint relation, but different scopes. If we collect together the scopes associated with a particular constraint relation we get a set of tuples which is itself a relation, but a relation over the set of variables, V. If we do this for each constraint relation occurring in our problem, we obtain a collection of such relations over V, which can be viewed as a relational structure $\mathbf{A}$ with universe V. Note that each relation E in $\mathbf{A}$ corresponds to a relation R in $\mathbf{B}$ of the same arity, and vice versa. This is captured in standard algebraic terminology by saying that the two relational structures, $\mathbf{A}$ and $\mathbf{B}$ are *similar*. Note also that a solution to the original CSP instance is a mapping from V to D that maps any tuple of variables related by a relation E in $\mathbf{A}$ to a tuple of values which are related by the corresponding relation R in $\mathbf{B}$. This is captured in standard algebraic terminology by saying that a solution is a *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$.

These observations gives rise to our third alternative formulation of the constraint satisfaction problem.

Definition 2.6. *An instance of the constraint satisfaction problem (CSP) is given by a pair of similar relational structures* $\mathbf{A}$ *and* $\mathbf{B}$*. The question is whether there exists a homomorphism from* $\mathbf{A}$ *to* $\mathbf{B}$*.*

This clean algebraic formulation of constraint satisfaction was introduced by Feder and Vardi [16] (and independently in [25]) and has turned out to be very useful for the analysis of the complexity of different forms of the problem.

Example 2.7 (Graph Homomorphism). The standard GRAPH HOMOMORPHISM problem consists in determining whether it is possible to map the vertices of a given graph G to the vertices of another given graph H so that adjacent vertices of G are mapped to adjacent vertices of H.

This can be viewed as a constraint satisfaction problem by viewing G and H as similar relational structures, each with a single binary relation. A homomorphism between these structures is precisely a mapping with the desired properties.

Example 2.8 (Graph Colouring). The standard GRAPH k-COLOURABILITY problem described in Example 2.3 can be viewed as the constraint satisfaction problem which asks whether there is a homomorphism from the given graph G to the structure $\mathbf{B}_{kCOL}$, defined in Example 2.3, which corresponds to a complete graph on k vertices.

This formulation of the problem makes it easy to see that the GRAPH k-COLOURABILITY problem is a special case of GRAPH HOMOMORPHISM.

Example 2.9 (Clique). The standard k-CLIQUE problem consists in determining whether a given graph G has a clique of size k, that is, a set of k vertices which are fully connected. This can be viewed as a constraint satisfaction problem which asks whether there is a homomorphism from the complete graph on k vertices to the given graph G.

This formulation of the problem makes it easy to see that k-CLIQUE is a special case of GRAPH HOMOMORPHISM.

3 Restricted Forms of CSP

It is clear from the examples in the previous sections that the general CSP is at least NP-hard. This has prompted many researchers to investigate the ways in which restricting the problem can reduce its complexity. We will call a restricted version of the CSP *tractable* if there is a polynomial-time algorithm to determine whether any instance of the restricted problem has a solution.

The algebraic formulation of the CSP, given in Definition 2.6, clearly identifies two separate aspects of the specification of an instance: the *source* structure, $\mathbf{A}$, and the *target* structure, $\mathbf{B}$.

Definition 3.1. *Given classes of structures, $\mathcal{A}$ and $\mathcal{B}$, we define the problem* $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$ *to be the class of CSP instances* $(\mathbf{A}, \mathbf{B})$*, where* $\mathbf{A} \in \mathcal{A}$ *and* $\mathbf{B} \in \mathcal{B}$*.*

The source structure $\mathbf{A}$ specifies the *scopes* of the constraints in Definition 2.4, so if we restrict the possible source structures that we allow in an instance, then we are restricting the set of variables and the ways in which the constraints may be imposed on those variables.

The target structure $\mathbf{B}$ specifies the constraint *relations* in Definition 2.4, so if we restrict the possible target structures that we allow in an instance, then we are restricting the set of possible values and the types of constraints that may be imposed on those values.

If $\mathcal{B}$ is the class of all structures, we write $\mathrm{CSP}(\mathcal{A}, -)$ in place of $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$. In this case we impose no restriction on the type of constraint, but some restriction on how the constraints may overlap. Such restrictions are known as *structural* restrictions, and have been widely studied [11,14,15,17,22,23,24,35]. One example of a constraint satisfaction problem with restricted structure is the k-Clique problem, described in Example 2.9, which is tractable for any bounded k, but NP-complete if k is unbounded. In general, the structural restrictions that ensure tractability are those that enforce a bound on some measure of width in the class of source structures allowed [22] (although the picture is more complicated if there is no fixed bound on the arity of the constraints [21]).

If $\mathcal{A}$ is the class of all structures, we write $\mathrm{CSP}(-, \mathcal{B})$ in place of $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$. In this case we impose no restriction on the way the constraints are placed, but some restriction on the forms of constraints that may be imposed. Such restrictions are known as *constraint language* restrictions, and have also been widely studied [2,8,9,10,16,29,28]. One example of a constraint satisfaction problem with restricted constraint language is the Graph k-Colourability problem, described in Example 2.3, which is tractable when $k \leq 2$, but NP-complete for $k \geq 3$. In the remainder of this paper we will focus on techniques to determine the complexity of the problems obtained by imposing constraint language restrictions.

Of course it is possible to impose other kinds of restrictions on the CSP, by restricting the possible pairs $(\mathbf{A}, \mathbf{B})$ that are allowed in instances in some other way. Such restrictions are sometimes referred to as *hybrid* restrictions [36], because they involve simultaneous restrictions on both the source structure and the target structure. Hybrid restrictions have been much less widely studied, although some interesting cases have recently been identified [12,38].

4 Constraint Languages, Expressive Power, and Reductions

From now on we shall focus on problems of the form $\mathrm{CSP}(-, \mathcal{B})$. To simplify the discussion, we shall assume that the class of structures $\mathcal{B}$ contains all structures with universe D and relations chosen from $\mathcal{L}$, for some fixed set D (with $|D| \geq 2$) and some fixed set $\mathcal{L}$ of relations over D. We will call D the domain and $\mathcal{L}$ the constraint language for our problem, and we will write the problem as $\mathrm{CSP}(\mathcal{L})$, rather than as $\mathrm{CSP}(-, \mathcal{B})$. Note that the domain D and the constraint language $\mathcal{L}$ may each be either finite or infinite.

Different choices of constraint language $\mathcal{L}$ give rise to a wide range of different problems, as the following examples indicate:

Example 4.1. If $\mathcal{L}_{LIN}$ denotes the set of all relations defined by linear equations over $\mathbb{Q}$, then $\mathrm{CSP}(\mathcal{L}_{LIN})$ is the problem of solving linear equations over $\mathbb{Q}$ described in Example 1.1.

Example 4.2. If $\mathcal{L}_{kCOL}$ denotes the set $\{\neq_k\}$ containing the single binary disequality relation over the set $\{1, 2, \ldots, k\}$, defined in Example 2.3, then $\mathrm{CSP}(\mathcal{L}_{kCOL})$ is the Graph k-Colourability problem described in Example 2.3.

Example 4.3. If $\mathcal{L}_{3SAT}$ denotes the set $\{R_1, R_2, \ldots, R_8\}$ consisting of all relations over the set $\{0,1\}$ which are defined by a single ternary clause, as described in Example 2.2, then CSP$(\mathcal{L}_{3SAT})$ is the 3-Sat problem described in Example 2.2.

Example 4.4. If $\mathcal{L}_{1in3SAT}$ denotes the set $\{T\}$ containing the single relation $T = \{\langle 1,0,0 \rangle, \langle 0,1,0 \rangle, \langle 0,0,1 \rangle\}$, then CSP$(\mathcal{L}_{1in3SAT})$, corresponds precisely to the One-In-Three Satisfiability problem [39], which is NP-complete.

Example 4.5. If $\mathcal{L}_{Interval}$ denotes the set $\{R_{\leq 1}\}$ containing the single relation

$$R_{\leq 1} = \{\langle a, b \rangle : a, b \in \mathbb{Q},\ a - b \leq 1\},$$

then one element of CSP$(\mathcal{L}_{Interval})$ is the CSP instance $\mathcal{P}$, which has 4 constraints, $\{C_1, C_2, C_3, C_4\}$, defined as follows:

- $C_1 = (\langle v_1, v_2 \rangle, R_{\leq 1})$;
- $C_2 = (\langle v_2, v_3 \rangle, R_{\leq 1})$;
- $C_3 = (\langle v_3, v_2 \rangle, R_{\leq 1})$;
- $C_4 = (\langle v_3, v_4 \rangle, R_{\leq 1})$.

The structure of this problem is illustrated in Figure 1.

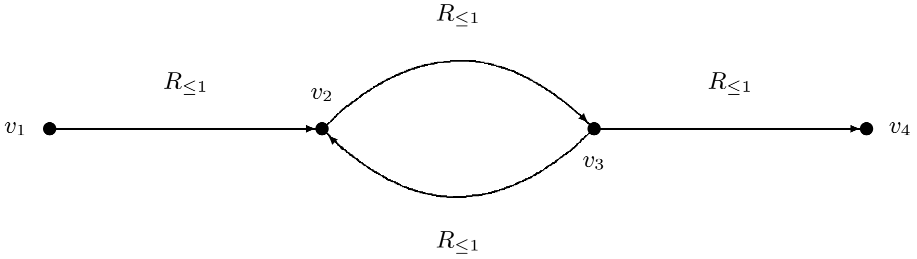


Fig. 1. The CSP defined in Example 4.5

Note that for any problem in CSP$(\mathcal{L})$, the explicit constraint relations must be elements of $\mathcal{L}$, but there may be implicit restrictions on some subsets of the variables for which the corresponding relations are not elements of $\mathcal{L}$, as the next example indicates.

Example 4.6. Reconsider the relation $R_{\leq 1}$ and the instance $\mathcal{P}$ from Example 4.5.

Note that there is no explicit constraint in $\mathcal{P}$ on the pair $\langle v_1, v_3 \rangle$. However, it is clear that the possible pairs of values which can be taken by this pair of variables are precisely the elements of the relation $R_{\leq 2} = \{\langle a, b \rangle : a, b \in \mathbb{Q}, a - b \leq 2\}$.

Similarly, the pairs of values which can be taken by the pair of variables $\langle v_1, v_4 \rangle$ are precisely the elements of the relation $R_{\leq 3} = \{\langle a, b \rangle : a, b \in \mathbb{Q}, a - b \leq 3\}$.

Finally, note that there are two constraints on the pair of variables $\langle v_2, v_3 \rangle$. The possible pairs of values which can be taken by this pair of variables are precisely the elements of the relation $S = \{\langle a, b \rangle : a, b \in \mathbb{Q}, -1 \leq (a - b) \leq 1\}$.

A natural way to capture these observations is to say that the relations $R_{\leq 2}, R_{\leq 3}$ and S can all be "expressed" using the relation $R_{\leq 1}$.

Definition 4.7. *A relation R can be **expressed** in a constraint language $\mathcal{L}$ if there exists an instance $\mathcal{P}$ in* $\mathrm{CSP}(\mathcal{L})$*, and a list, s, of variables, such that the solutions to $\mathcal{P}$ when restricted to s give precisely the tuples of R.*

For any constraint language $\mathcal{L}$, the set of all relations which can be expressed in $\mathcal{L}$ will be called the **expressive power** of $\mathcal{L}$, and will be denoted $\mathcal{L}^+$.

Example 4.8. By generalising the constructions given in Example 4.6, it is clear that when $\mathcal{L} = \{R_{\leq 1}\}$, $\mathcal{L}^+$ contains all binary relations of the form

$$\{\langle a, b \rangle \,:\, a, b \in \mathbb{Q},\ -n_1 \leq (a - b) \leq n_2\}$$

where n_1 and n_2 are arbitrary positive integers, or ∞, as well as relations of all other arities.

The next result is simple but fundamental: it shows that if we can express a relation in a constraint language, then we can add it to the language without changing the complexity of the associated class of problems.

Proposition 4.9 ([25]). *For any constraint language $\mathcal{L}$ and any relation R which can be expressed in $\mathcal{L}$,* $\mathrm{CSP}(\mathcal{L} \cup \{R\})$ *is reducible to* $\mathrm{CSP}(\mathcal{L})$ *in linear time and logarithmic space.*

Corollary 4.10. *For any constraint language $\mathcal{L}$, and any finite constraint language $\mathcal{L}_0$, if $\mathcal{L}_0 \subseteq \mathcal{L}^+$, then* $\mathrm{CSP}(\mathcal{L}_0)$ *is reducible to* $\mathrm{CSP}(\mathcal{L})$ *in log space.*

Hence the notion of expressive power provides a powerful general tool for finding reductions, and hence comparing and analysing different forms of constraint satisfaction problems over different constraint languages.

5 Calculating Expressive Power

Definition 4.7 states that a relation R can be expressed in a language $\mathcal{L}$ if there is some problem in $\mathrm{CSP}(\mathcal{L})$ which imposes that relation on some of its variables. Taking the logical view of the CSP given in Definition 2.1, this means that a relation can be expressed in a language $\mathcal{L}$ if it can be defined by a primitive positive formula over $\mathcal{L}$. The question of what can be defined by a primitive positive formula is a fundamental question in universal algebra, and a useful answer was obtained for relations over a finite domain in the 1960's [19,6]. This result was recently extended to relations over a countably infinite domain that are ω-categorical[1] [4]. The answer in both cases involves certain algebraic operations known as *polymorphisms*, which are defined as follows:

Definition 5.1. *An operation $f : D^k \to D$ is a **polymorphism** of an n-ary relation R if the tuple $(f(t_1[1], \ldots, t_k[1]), \ldots, f(t_1[n], \ldots, t_k[n])) \in R$ for all choices of tuples $t_1, \ldots, t_k \in R$.*

[1] A countably infinite structure is called ω-categorical if all countable models of its first-order theory are isomorphic. An important example is the structure $(\mathbb{Q}, <)$.

An operation f is said to be a polymorphism of a relational structure $\mathbf{A}$ if f is a polymorphism of every relation of $\mathbf{A}$. In fact, it can be shown that the k-ary polymorphisms of a structure $\mathbf{A}$ are precisely the homomorphisms from $\mathbf{A}^k$ to $\mathbf{A}$ for the *product structure* $\mathbf{A}^k$.

Theorem 5.2 ([19,6,4]). *Let $\mathbf{A}$ be a finite or ω-categorical structure. A relation R has a primitive-positive definition over $\mathbf{A}$ if and only if every polymorphism of $\mathbf{A}$ is a polymorphism of R.*

For any constraint language $\mathcal{L}$ over a finite domain D, Theorem 5.2 tells us that the expressive power of $\mathcal{L}$ is determined by the polymorphisms of $\mathcal{L}$, which can themselves be seen as solutions to a constraint satisfaction problem in CSP($\mathcal{L}$).

A straightforward consequence of this result is that, for any language over a finite domain, there is a "universal construction" which can be used to determine whether a relation is expressible in that language.

Definition 5.3. *Let $\mathcal{L}$ be a set of relations over a finite set D.*

For any natural number $m > 0$, the indicator problem *for $\mathcal{L}$ of order m is defined to be the CSP instance $\mathcal{IP}(\mathcal{L}, m)$ with set of variables D^m, each with domain D, and constraints $\{C_1, C_2, \ldots, C_q\}$, where $q = \sum_{R\in\mathcal{L}} |R|^m$. For each $R \in \mathcal{L}$, and for each sequence $t_1, t_2, \ldots, t_m$ of tuples from R, there is a constraint $C_i = \langle s_i, R \rangle$ with $s_i = \langle v_1, v_2, \ldots, v_n \rangle$, where n is the arity of R and $v_j = \langle t_1[j], t_2[j], \ldots, t_m[j] \rangle$ for $j = 1$ to n.*

Note that for any set of relations $\mathcal{L}$ over a set D, $\mathcal{IP}(\mathcal{L}, m)$ has $|D|^m$ variables, and each variable corresponds to an m-tuple over D. A concrete example of an indicator problem is given below, and more examples can be found in [27]. It is shown in [28] that the solutions to $\mathcal{IP}(\mathcal{L}, m)$ are precisely the polymorphisms of $\mathcal{L}$ of arity m, and hence, by Theorem 5.2, we have:

Corollary 5.4. *Let $\mathcal{L}$ be a set of relations over a finite set D, let $R = \{t_1, \ldots, t_m\}$ be any relation over D, and let n be the arity of R.*

The relation R can be expressed in the language $\mathcal{L}$ if and only if the tuples of R are given by the solutions to $\mathcal{IP}(\mathcal{L}, m)$ restricted to the variables[2] *$v_1, v_2, \ldots, v_n$, where $v_j = \langle t_1[j], t_2[j], \ldots, t_m[j] \rangle$ for $j = 1$ to n.*

Example 5.5. Let $\mathcal{L}$ be the set containing the single binary relation, $R_\times$, over the set $D = \{0, 1, 2\}$, defined as follows:

$$R_\times = \{\langle 0,0 \rangle, \langle 0,1 \rangle, \langle 1,0 \rangle, \langle 1,2 \rangle, \langle 2,1 \rangle, \langle 2,2 \rangle\}$$

The indicator problem for $\{R_\times\}$ of order 2, $\mathcal{IP}(\{R_\times\}, 2)$, has 9 variables and 36 constraints. The set of variables is

$$\{\langle 0,0 \rangle, \langle 0,1 \rangle, \langle 0,2 \rangle, \langle 1,0 \rangle, \langle 1,1 \rangle, \langle 1,2 \rangle, \langle 2,0 \rangle, \langle 2,1 \rangle, \langle 2,2 \rangle\},$$

and the set of constraints is

[2] This list of variables is not always uniquely defined - re-ordering the elements of R may give a different list, but the same result holds for all such lists.

$$\begin{aligned}\{\ &((\langle 0,0\rangle,\langle 0,0\rangle),R_\times),\ ((\langle 0,0\rangle,\langle 0,1\rangle),R_\times),\ ((\langle 0,0\rangle,\langle 1,0\rangle),R_\times),\ ((\langle 0,0\rangle,\langle 1,1\rangle),R_\times),\\ &((\langle 0,1\rangle,\langle 0,0\rangle),R_\times),\ ((\langle 0,1\rangle,\langle 0,2\rangle),R_\times),\ ((\langle 0,1\rangle,\langle 1,0\rangle),R_\times),\ ((\langle 0,1\rangle,\langle 1,2\rangle),R_\times),\\ &((\langle 0,2\rangle,\langle 0,1\rangle),R_\times),\ ((\langle 0,2\rangle,\langle 0,2\rangle),R_\times),\ ((\langle 0,2\rangle,\langle 1,1\rangle),R_\times),\ ((\langle 0,2\rangle,\langle 1,2\rangle),R_\times),\\ &((\langle 1,0\rangle,\langle 0,0\rangle),R_\times),\ ((\langle 1,0\rangle,\langle 0,1\rangle),R_\times),\ ((\langle 1,0\rangle,\langle 2,0\rangle),R_\times),\ ((\langle 1,0\rangle,\langle 2,1\rangle),R_\times),\\ &((\langle 1,1\rangle,\langle 0,0\rangle),R_\times),\ ((\langle 1,1\rangle,\langle 0,2\rangle),R_\times),\ ((\langle 1,1\rangle,\langle 2,0\rangle),R_\times),\ ((\langle 1,1\rangle,\langle 2,2\rangle),R_\times),\\ &((\langle 1,2\rangle,\langle 0,1\rangle),R_\times),\ ((\langle 1,2\rangle,\langle 0,2\rangle),R_\times),\ ((\langle 1,2\rangle,\langle 2,1\rangle),R_\times),\ ((\langle 1,2\rangle,\langle 2,2\rangle),R_\times),\\ &((\langle 2,0\rangle,\langle 1,0\rangle),R_\times),\ ((\langle 2,0\rangle,\langle 1,1\rangle),R_\times),\ ((\langle 2,0\rangle,\langle 2,0\rangle),R_\times),\ ((\langle 2,0\rangle,\langle 2,1\rangle),R_\times),\\ &((\langle 2,1\rangle,\langle 1,0\rangle),R_\times),\ ((\langle 2,1\rangle,\langle 1,2\rangle),R_\times),\ ((\langle 2,1\rangle,\langle 2,0\rangle),R_\times),\ ((\langle 2,1\rangle,\langle 2,2\rangle),R_\times),\\ &((\langle 2,2\rangle,\langle 1,1\rangle),R_\times),\ ((\langle 2,2\rangle,\langle 1,2\rangle),R_\times),\ ((\langle 2,2\rangle,\langle 2,1\rangle),R_\times),\ ((\langle 2,2\rangle,\langle 2,2\rangle),R_\times)\ \}.\end{aligned}$$

This problem has 32 solutions, which may be expressed in tabular form as follows:

	Variables								
	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,2\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$	$\langle 1,2\rangle$	$\langle 2,0\rangle$	$\langle 2,1\rangle$	$\langle 2,2\rangle$
Solution 1	0	0	0	0	0	0	0	0	0
Solution 2	0	0	0	0	0	0	0	1	0
Solution 3	0	0	0	0	0	1	0	0	0
Solution 4	0	0	0	0	1	0	0	0	0
Solution 5	0	0	0	0	1	0	0	1	0
Solution 6	0	0	0	0	1	1	0	0	0
Solution 7	0	0	0	1	0	0	0	0	0
Solution 8	0	0	0	1	0	1	0	0	0
Solution 9	0	0	0	1	1	0	0	0	0
Solution 10	0	0	0	1	1	1	0	0	0
Solution 11	0	0	0	1	1	1	2	2	2
Solution 12	0	1	0	0	0	0	0	0	0
Solution 13	0	1	0	0	0	0	0	1	0
Solution 14	0	1	0	0	1	0	0	0	0
Solution 15	0	1	0	0	1	0	0	1	0
Solution 16	0	1	2	0	1	2	0	1	2
Solution 17	2	1	0	2	1	0	2	1	0
Solution 18	2	1	2	2	1	2	2	1	2
Solution 19	2	1	2	2	1	2	2	2	2
Solution 20	2	1	2	2	2	2	2	1	2
Solution 21	2	1	2	2	2	2	2	2	2
Solution 22	2	2	2	1	1	1	0	0	0
Solution 23	2	2	2	1	1	1	2	2	2
Solution 24	2	2	2	1	1	2	2	2	2
Solution 25	2	2	2	1	2	1	2	2	2
Solution 26	2	2	2	1	2	2	2	2	2
Solution 27	2	2	2	2	1	1	2	2	2
Solution 28	2	2	2	2	1	2	2	1	2
Solution 29	2	2	2	2	1	2	2	2	2
Solution 30	2	2	2	2	2	1	2	2	2
Solution 31	2	2	2	2	2	2	2	1	2
Solution 32	2	2	2	2	2	2	2	2	2

By Corollary 5.4, to verify that the relation $R_1 = \{\langle 0,0\rangle, \langle 2,2\rangle\}$ can be expressed in the language $\{R_\times\}$, we simply have to check that the restriction of this set of solutions to the pair of (identical) variables $\langle 0,2\rangle$ and $\langle 0,2\rangle$ gives precisely the tuples $\langle 0,0\rangle$ and $\langle 2,2\rangle$ of R_1.

Conversely, to establish that the relation $R_2 = \{\langle 0,0\rangle, \langle 0,1\rangle\}$ *cannot* be expressed in the language $\{R_\times\}$, we simply observe that the restriction of this set of solutions to the variables $\langle 0,0\rangle$ and $\langle 0,1\rangle$ gives the tuples $\langle 0,0\rangle, \langle 0,1\rangle, \langle 2,1\rangle$ and $\langle 2,2\rangle$, which are *not* all contained in R_2. (The same result is obtained by restricting to the variables $\langle 0,0\rangle$ and $\langle 1,0\rangle$.)

6 Complexity

The results presented above have led to some very general techniques to identify reductions and hence determine the complexity of $\mathrm{CSP}(\mathcal{L})$.

Definition 6.1. *A constraint language, $\mathcal{L}$, is said to be* ***tractable*** *if* $\mathrm{CSP}(\mathcal{L}')$ *can be solved in polynomial time, for each finite subset $\mathcal{L}' \subseteq \mathcal{L}$.*

A constraint language, $\mathcal{L}$, is said to be ***NP-hard*** *if* $\mathrm{CSP}(\mathcal{L}')$ *is NP-hard, for some finite subset $\mathcal{L}' \subseteq \mathcal{L}$.*

It has recently been shown that *any* computational problem is polynomial-time Turing equivalent to a problem of the form $\mathrm{CSP}(\mathcal{L})$ for some constraint language $\mathcal{L}$ over an infinite domain D [3]. However, for any constraint language $\mathcal{L}$ over a *finite* domain D, $\mathrm{CSP}(\mathcal{L})$ lies in the complexity class NP, because a solution can be verified in polynomial time (assuming that membership in each relation of $\mathcal{L}$ can be verified in polynomial time).

By Proposition 4.9, we can show that many languages are NP-complete by simply showing that they can express some known NP-complete language.

Example 6.2. Recall the NP-complete language $\mathcal{L}_{1in3SAT}$ described in Example 4.4, consisting of a single relation T containing 3 tuples.

By Corollary 5.4, for any language $\mathcal{L}$, and any two domain elements d_0, d_1, if the solutions to $\mathcal{IP}(\mathcal{L}, 3)$, restricted to the variables $\langle\, d_1, d_0, d_0 \,\rangle, \langle\, d_0, d_1, d_0 \,\rangle$ and $\langle\, d_0, d_0, d_1 \,\rangle$ is equal to $\{\langle\, d_1, d_0, d_0 \,\rangle, \langle\, d_0, d_1, d_0 \,\rangle, \langle\, d_0, d_0, d_1 \,\rangle\}$, then $\mathcal{L}$ can express a relation isomorphic to T, and hence is NP-complete.

In particular, if $\mathcal{IP}(\mathcal{L}, 3)$ has only 3 solutions then $\mathcal{L}$ is NP-complete.

Note that this provides a purely mechanical procedure to establish NP-completeness of a constraint language, without having to design any specific reductions, or invent any new gadgets or constructions, as the following example illustrates.

Example 6.3. Reconsider the relation $R_\times$ over $\{0, 1, 2\}$, defined in Example 5.5.

The language $\{R_\times\}$ is clearly tractable, because any problem in $\mathrm{CSP}(\{R_\times\})$ has the trivial solution in which every variable takes the value 0.

However, if we consider the language $\mathcal{L}_0 = \{R_\times, R_0\}$, where $R_0 = \{\langle\, 0, 1, 2 \,\rangle\}$ then we find that the indicator problem for $\mathcal{L}_0$ of order 3, $\mathcal{IP}(\mathcal{L}_0, 3)$, with 27 variables and 217 constraints, has only 3 solutions. Hence, $\mathcal{L}_0$ is NP-complete.

For any constraint language $\mathcal{L}$, the indicator problem $\mathcal{IP}(\mathcal{L}, 3)$ has at least 3 solutions (all 3 projection operations are always solutions). Hence the only way that a language can fail to be NP-complete, is if it has additional solutions, or in other words, additional polymorphisms.

Example 6.4. The language $\mathcal{L}_{LIN}$ defined in Example 4.1 contains precisely those relations with the ternary polymorphism f defined by $f(x, y, z) = x - y + z$. In fact, having this single polymorphism is sufficient to ensure tractability for any constraint language over a (finite or infinite) field [28,2].

Example 6.5. Let $\mathcal{L}$ be any constraint language over the domain $\{0, 1\}$.

In this case CSP($\mathcal{L}$) corresponds exactly to the Generalised Satisfiability problem [18], and it is known that $\mathcal{L}$ is NP-complete unless it falls into one of the following 6 classes [39]:

Class 0a All relations contain the tuple $\langle 0, 0, \ldots, 0 \rangle$.
Class 0b All relations contain the tuple $\langle 1, 1, \ldots, 1 \rangle$.
Class Ia All relations can be defined using Horn clauses.
Class Ib All relations can be defined using anti-Horn clauses[3].
Class II All relations can be defined using clauses with at most 2 literals.
Class III All relations can be defined using linear equations over $\mathbb{Z}_2$.

The indicator problem for $\mathcal{L}$ of order 3, $\mathcal{IP}(\mathcal{L}, 3)$, has 8 variables, corresponding to the 8 possible Boolean sequences of length 3. It has 256 possible solutions, corresponding to the 256 possible assignments of Boolean values to these 8 variables.

Amongst these 256 possible solutions, we can identify 6 distinguished assignments as shown in the following table.

	Variables							
	$\langle 0,0,0 \rangle$	$\langle 0,0,1 \rangle$	$\langle 0,1,0 \rangle$	$\langle 0,1,1 \rangle$	$\langle 1,0,0 \rangle$	$\langle 1,0,1 \rangle$	$\langle 1,1,0 \rangle$	$\langle 1,1,1 \rangle$
Class 0a - Constant 0	0	0	0	0	0	0	0	0
Class 0b - Constant 1	1	1	1	1	1	1	1	1
Class Ia - Horn	0	0	0	0	0	0	0	1
Class Ib - Anti-Horn	0	1	1	1	1	1	1	1
Class II - 2-Sat	0	0	0	1	0	1	1	1
Class III - Linear	0	1	1	0	1	0	0	1

It can be shown [26] that the language $\mathcal{L}$ falls into one of the 6 tractable classes described above if and only if $\mathcal{IP}(\mathcal{L}, 3)$ has the corresponding solution, as shown in this table, and hence $\mathcal{L}$ has the corresponding polymorphism.

There has been considerable progress in identifying classes of polymorphisms which are sufficient to ensure tractability (see [9] for a recent survey). The set of polymorphisms of a constraint language is closed under composition, and can be viewed as the set of term operations of an *algebra* [8]; some deep algebraic ideas have been used to investigate the possible forms such a set can take [9], leading to a number of intriguing new formulations of the following *dichotomy conjecture* originally proposed in [16]:

Conjecture 6.6 ([8,9]). Let $\mathcal{L}$ be a constraint language over a finite domain, such that $\mathcal{L}$ contains all unary constant relations. If $\mathcal{L}$ has a polymorphism, f, that satisfies the identities $f(y, x, x, \ldots, x) = f(x, y, x, \ldots, x) = \ldots = f(x, x, \ldots, x, y)$, then it is tractable. Otherwise it is NP-complete.

Conjecture 6.6 is currently known to hold for domains of size 2 (Example 6.5) and 3 [10,9] and for languages over any finite domain containing a single binary symmetric relation [7] (see Example 2.7). Assuming that P≠NP, the condition stated in the conjecture is known to be a necessary condition for tractability [9]; it remains to be shown whether it is always sufficient.

[3] An *anti-Horn* clause is a disjunction of literals, with at most one negative literal.

References

1. Becker, T., Weispfenning, V.: Gröbner Bases: A Computational Approach to Commutative Algebra. Graduate Texts in Mathematics. Springer, Heidelberg (1993)
2. Bodirsky, M.: Constraint satisfaction problems with infinite templates. In: Creignou, N., et al. (eds.) Complexity of Constraints. LNCS, vol. 5250, pp. 196–228. Springer, Heidelberg (2008)
3. Bodirsky, M., Grohe, M.: Non-dichotomies in constraint satisfaction complexity. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 184–196. Springer, Heidelberg (2008)
4. Bodirsky, M., Nešetřil, J.: Constraint satisfaction with countable homogeneous templates. Journal of Logic and Computation 16, 359–373 (2006)
5. Bodirsky, M., Nordh, G., von Oertzen, T.: Integer programming with 2-variable equations and 1-variable inequalities. Inf. Proc. Letters 109, 572–575 (2009)
6. Bodnarchuk, V., Kaluzhnin, L., Kotov, V., Romov, B.: Galois theory for Post algebras. I. Cybernetics and Systems Analysis 5, 243–252 (1969)
7. Bulatov, A.: H-coloring dichotomy revisited. Theoretical Computer Science 349(1), 31–39 (2005)
8. Bulatov, A., Krokhin, A., Jeavons, P.: Classifying the complexity of constraints using finite algebras. SIAM Journal on Computing 34(3), 720–742 (2005)
9. Bulatov, A., Valeriote, M.: Recent results on the algebraic approach to the CSP. In: Creignou, N., et al. (eds.) Complexity of Constraints. LNCS, vol. 5250, pp. 68–92. Springer, Heidelberg (2008)
10. Bulatov, A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. Journal of the ACM 53(1), 66–120 (2006)
11. Cohen, D., Jeavons, P., Gyssens, M.: A unified theory of structural tractability for constraint satisfaction problems. Journal of Computer and System Sciences 74, 721–743 (2007)
12. Cooper, M., Jeavons, P., Salamon, A.: Hybrid tractable CSPs which generalize tree structure. In: ECAI 2008, Proceedings of the 18th European Conference on Artificial Intelligence, Patras, Greece, July 21–25, 2008, pp. 530–534. IOS Press, Amsterdam (2008)
13. Creignou, N., Khanna, S., Sudan, M.: Complexity Classification of Boolean Constraint Satisfaction Problems. SIAM Monographs on Discrete Mathematics and Applications, vol. 7. Society for Industrial and Applied Mathematics, Philadelphia (2001)
14. Dalmau, V., Kolaitis, P., Vardi, M.: Constraint satisfaction, bounded treewidth, and finite-variable logics. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 310–326. Springer, Heidelberg (2002)
15. Dechter, R., Pearl, J.: Tree clustering for constraint networks. Artificial Intelligence 38, 353–366 (1989)
16. Feder, T., Vardi, M.: The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. SIAM Journal on Computing 28, 57–104 (1998)
17. Freuder, E.: A sufficient condition for backtrack-bounded search. Journal of the ACM 32, 755–761 (1985)
18. Garey, M., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)

19. Geiger, D.: Closed systems of functions and predicates. Pacific Journal of Mathematics 27, 95–100 (1968)
20. Green, M., Jefferson, C.: Structural tractability of propagated constraints. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 372–386. Springer, Heidelberg (2008)
21. Grohe, M.: The structure of tractable constraint satisfaction problems. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 58–72. Springer, Heidelberg (2006)
22. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. Journal of the ACM 54, 1–24 (2007)
23. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. In: SODA 2006: Proceedings of the 17th annual ACM-SIAM Symposium on Discrete Algorithm, pp. 289–298. ACM Press, New York (2006)
24. Gyssens, M., Jeavons, P., Cohen, D.: Decomposing constraint satisfaction problems using database techniques. Artificial Intelligence 66(1), 57–89 (1994)
25. Jeavons, P.: On the algebraic structure of combinatorial problems. Theoretical Computer Science 200, 185–204 (1998)
26. Jeavons, P., Cohen, D.: An algebraic characterization of tractable constraints. In: Li, M., Du, D.-Z. (eds.) COCOON 1995. LNCS, vol. 959, pp. 633–642. Springer, Heidelberg (1995)
27. Jeavons, P., Cohen, D., Gyssens, M.: A test for tractability. In: Freuder, E.C. (ed.) CP 1996. LNCS, vol. 1118, pp. 267–281. Springer, Heidelberg (1996)
28. Jeavons, P., Cohen, D., Gyssens, M.: Closure properties of constraints. Journal of the ACM 44, 527–548 (1997)
29. Jeavons, P., Cooper, M.: Tractable constraints on ordered domains. Artificial Intelligence 79(2), 327–339 (1995)
30. Jefferson, C., Jeavons, P., Green, M., van Dongen, M.: Representing and solving finite-domain constraint problems using systems of polynomials. Technical Report RR-07-08, Oxford University Computing Laboratory (2007)
31. Jonsson, P., Krokhin, A.: Recognizing frozen variables in constraint satisfaction problems. Theoretical Computer Science 329(1-3), 93–113 (2004)
32. Larose, B., Zádori, L.: Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. International Journal of Algebra and Computation 16, 563–582 (2006)
33. Lenstra, H.: Integer programming with a fixed number of variables. Mathematics of Operations Research 8, 538–548 (1983)
34. Matijasevič, J., Robinson, J.: Reduction of an arbitrary Diophantine equation to one in 13 unknowns. Acta Arithmeticae 27, 521–553 (1975)
35. Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Information Sciences 7, 95–132 (1974)
36. Pearson, J., Jeavons, P.: A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, Royal Holloway, University of London (July 1997)
37. Rossi, F., van Beek, P., Walsh, T. (eds.): The Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
38. Salamon, A., Jeavons, P.: Perfect constraints are tractable. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 524–528. Springer, Heidelberg (2008)
39. Schaefer, T.: The complexity of satisfiability problems. In: Proceedings 10th ACM Symposium on Theory of Computing, STOC 1978, pp. 216–226 (1978)
40. Simonis, H.: Sudoku as a constraint problem. In: CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems, pp. 13–27 (2005)

On the Use of Automata for Deciding Linear Arithmetic

Pierre Wolper

Université de Liège, Belgium

Abstract of Invited Talk

This talk presents a survey of automata-based techniques for representing and manipulating linear arithmetic constraints. After introducing the basic concepts used in this approach, both representing integer constraints by finite-word automata and real constraints by infinite-word automata is discussed. Various results about the construction of automata from constraints and about the specific properties of automata representing arithmetic constraints are then presented. Finally, it is shown how this approach leads to simple and natural decision procedures that are in some ways related to tableaux.

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, p. 16, 2009.

Comparative Concept Similarity over Minspaces: Axiomatisation and Tableaux Calculus

Régis Alenda[1], Nicola Olivetti[1], and Camilla Schwind[2]

[1] LSIS - UMR CNRS 6168,
Domaine Universitaire de Saint-Jérôme, Avenue Escadrille Normandie-Niemen,
13397 MARSEILLE CEDEX 20
regis.alenda@lsis.org, nicola.olivetti@univ-cezanne.fr
[2] LIF - UMR CNRS 6166,
Centre de Mathématiques et Informatique,
39 rue Joliot-Curie - F-13453 Marseille Cedex13
camilla.schwind@lif.univ-mrs.fr

Abstract. We study the logic of comparative concept similarity $\mathcal{CSL}$ introduced by Sheremet, Tishkovsky, Wolter and Zakharyaschev to capture a form of qualitative similarity comparison. In this logic we can formulate assertions of the form "objects A are more similar to B than to C". The semantics of this logic is defined by structures equipped with distance functions evaluating the similarity degree of objects. We consider here the particular case of the semantics induced by *minspaces*, the latter being distance spaces where the minimum of a set of distances always exists. It turns out that the semantics over arbitrary minspaces can be equivalently specified in terms of preferential structures, typical of conditional logics. We first give a direct axiomatisation of this logic over Minspaces. We next define a decision procedure in the form of a tableaux calculus. Both the calculus and the axiomatisation take advantage of the reformulation of the semantics in terms of preferential structures.

1 Introduction

The logics of comparative concept similarity $\mathcal{CSL}$ have been introduced in [9] to capture a form of qualitative comparison between concept instances. In these logics we can express assertions or judgments of the form: "Renault Clio is more similar to Peugeot 207 than to WW Golf". These logics may find an application in ontology languages, whose logical base is provided by Description Logics (DL), allowing concept definitions based on proximity/similarity measures. For instance ([9]), the color "Reddish" may be defined as a color which is more similar to a prototypical "Red" than to any other color (in some color model as RGB). The aim is to dispose of a language in which logical classification provided by standard DL is integrated with classification mechanisms based on calculation of proximity measures. The latter is typical for instance of domains like bio-informatics or linguistics. In a series of papers [9,11,5,10] the authors propose several languages comprising absolute similarity measures and comparative similarity operator(s). In this paper we consider a logic $\mathcal{CSL}$ obtained by

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 17–31, 2009.

adding to a propositional language just one binary modal connective $\leftleftarrows$ expressing comparative similarity. In this language the above examples can be encoded (using a description logic notation) by:

(1) $\mathit{Reddish} \equiv \{\mathit{Red}\} \leftleftarrows \{\mathit{Green}, \ldots, \mathit{Black}\}$,
(2) $\mathit{Clio} \sqsubseteq (\mathit{Peugeot207} \leftleftarrows \mathit{Golf})$.

In a more general setting, the language might contain several $\leftleftarrows_{\mathrm{Feature}}$ where each Feature corresponds to a specific distance function d_{Feature} measuring the similarity of objects with respect to one Feature (size, price, power, taste, color...). In our setting a KB about cars may collect assertions of the form (2) and others, say:

(3) $\mathit{Clio} \sqsubseteq (\mathit{Golf} \leftleftarrows \mathit{Ferrari430})$,
(4) $\mathit{Clio} \sqsubseteq (\mathit{Peugeot207} \leftleftarrows \mathit{MaseratiQP})$,

together with some general axioms for classifying cars:

$\mathit{Peugeot207} \sqsubseteq \mathit{Citycar}$,
$\mathit{SportLuxuryCar} \equiv \mathit{MaseratiQP} \sqcup \mathit{Ferrari430}$.

Comparative similarity assertions such as (2)–(4) might not necessarily be the fruit of an objective numerical calculation of similarity measures, but they could be determined just by the (integration of) subjective opinions of agents, answering, for instance, to questions like: "Is Clio more similar to Golf or to Ferrari 430?". In any case, the logic $\mathcal{CSL}$ allows one to perform some kind of reasoning, for instance the following conclusions will be supported:

$\mathit{Clio} \sqsubseteq (\mathit{Peugeot207} \leftleftarrows \mathit{Ferrari430})$,
$\mathit{Clio} \sqsubseteq (\mathit{Citycar} \leftleftarrows \mathit{SportLuxuryCar})$.

and also $\mathit{Clio} \sqsubseteq (\mathit{Citycar} \leftleftarrows \mathit{SportLuxuryCar} \sqcap \mathit{4Wheels})$.

The semantics of $\mathcal{CSL}$ is defined in terms of distance spaces, that is to say structures equipped by a distance function d, whose properties may vary according to the logic under consideration. In this setting, the evaluation of $A \leftleftarrows B$ can be informally stated as follows: x satisfies $A \leftleftarrows B$ iff $d(x, A) < d(x, B)$ meaning that the object x is an instance of the concept $A \leftleftarrows B$ (i.e. it belongs to things that are more similar to A than to B) if x is strictly closer to A-objects than to B-objects according to distance function d, where the distance of an object to a set of objects is defined as the *infimum* of the distances to each object in the set.

In [9,11,5,10], the authors have investigated the logic $\mathcal{CSL}$ with respect to different classes of distance models, see [11] for a survey of results about decidability, complexity, expressivity, and axiomatisation. Remarkably it is shown that $\mathcal{CSL}$ is undecidable over subspaces of the reals. Moreover $\mathcal{CSL}$ over arbitrary distance spaces can be seen as a fragment, indeed a powerful one (including for instance the logic $\mathbf{S4}_u$ of topological spaces), of a general logic for spatial reasoning comprising different modal operators defined by (bounded) quantified distance expressions.

The authors have pointed out that in case the distance spaces are assumed to be *minspaces*, that is spaces where the infimum of a set of distances is actually their *minimum*, the logic $\mathcal{CSL}$ is naturally related to some conditional logics. The semantics of the latter is often expressed in terms of preferential structures, that is to say possible-world structures equipped by a family of strict partial (pre)-orders $\prec_x$ indexed on objects/worlds [7,12]. The intended meaning of the relation $y \prec_x z$ is namely that x is more similar to y than to z. It is not hard to see that the semantics over minspaces is equivalent to the semantics over preferential structures satisfying the well-known principle of Limit Assumption according to which the set of minimal elements of a non-empty set always exists.

The minspace property entails the restriction to spaces where the distance function is discrete. This requirement does not seem incompatible with the purpose of representing qualitative similarity comparisons, whereas it might not be reasonable for applications of $\mathcal{CSL}$ to spatial reasoning.

In this paper we contribute to the study of $\mathcal{CSL}$ over minspaces. We first show (unsurprisingly) that the semantics of $\mathcal{CSL}$ on minspaces can be equivalently restated in terms of preferential models satisfying some additional conditions, namely modularity, centering, and the limit assumption. We then give a *direct* axiomatization of this logic. This problem was not considered in detail in [11]. In that paper an axiomatization of $\mathcal{CSL}$ over arbitrary distance models is proposed, but it makes use of an additional operator. Our axiomatisation is simpler and only employs $\leftleftarrows$. Next, we define a tableaux calculus for checking satisfiability of $\mathcal{CSL}$ formulas. Our tableaux procedure makes use of labelled formulas and pseudo-modalities indexed on worlds $\Box_x$, similarly to the calculi for conditional logics defined in [3]. Termination is assured by suitable blocking conditions. To the best of our knowledge our calculus provides the first known practically-implementable decision procedure for $\mathcal{CSL}$ logic.

2 The Logic of *Comparative Concept Similarity* $\mathcal{CSL}$

The language $\mathcal{L}_{\mathcal{CSL}}$ of $\mathcal{CSL}$ is generated from a (countable) set of propositional variables $V_1, V_2, \ldots \in \mathcal{V}_p$ by ordinary propositional connectives plus $\leftleftarrows$:
$A, B \ ::= \ \bot \mid V_i \mid \neg A \mid A \sqcap B \mid A \leftleftarrows B$ (where $V_i \in \mathcal{V}_p$).

The semantics of $\mathcal{CSL}$ introduced in [9] makes use of *distance spaces* in order to represent the similarity degree between objects. A distance space is a pair (Δ, d) where Δ is a non-empty set, and $d : \Delta \times \Delta \rightarrow \mathbb{R}^{\geq 0}$ is a *distance function* satisfying the following condition:

$$\text{(ID)} \qquad \forall x, y \in \Delta, \quad d(x, y) = 0 \text{ iff } x = y \ .$$

Two further properties are usually considered: symmetry and triangle inequality. We briefly discuss them below.

The distance between an object w and a non-empty subset X of Δ is defined by $d(w, X) = \inf\{d(w, x) \mid x \in X\}$. If $X = \emptyset$, then $d(w, X) = \infty$. If for every object w and for every (non-empty) subset X we have the following property

$$\text{(MIN)} \qquad \inf\{d(w, x) \mid x \in X\} = \min\{d(w, x) \mid x \in X\} \ ,$$

we say that (Δ, d) is a *minspace.*

We next define $\mathcal{CSL}$-distance models as Kripke models based on distance spaces:

Definition 1 ($\mathcal{CSL}$-distance model)
A $\mathcal{CSL}$-distance model is a triple $\mathcal{M} = (\Delta, d, .^{\mathcal{M}})$ where:

- *Δ is a non-empty set of* objects.
- *d is a distance on Δ (so that (Δ, d) is a distance space).*
- *$.^{\mathcal{M}} : \mathcal{V}_p \to 2^{\Delta}$ is the* evaluation function *which assigns to each propositional variable V_i a set $V_i^{\mathcal{M}} \subseteq \Delta$. We further stipulate:*
  $\bot^{\mathcal{M}} = \emptyset$, $\quad (\neg C)^{\mathcal{M}} = \Delta - C^{\mathcal{M}}$, $\quad (C \sqcap D)^{\mathcal{M}} = C^{\mathcal{M}} \cap D^{\mathcal{M}}$,
  $(C \Leftarrow D)^{\mathcal{M}} = \left\{w \in \Delta \;\middle|\; d(w, C^{\mathcal{M}}) < d(w, D^{\mathcal{M}})\right\}$.

*If (Δ, d) is a minspace, $\mathcal{M}$ is called a $\mathcal{CSL}$-*distance minspace model *(or simply a* minspace model*). We say that a formula A is* valid in a model $\mathcal{M}$ *if $A^{\mathcal{M}} = \Delta$. We say that a formula A is* valid *if A is valid in every $\mathcal{CSL}$-distance model.*

As mentioned above, the distance function might be required to satisfy the further conditions of symmetry (SYM) $(d(x, y) = d(y, x))$ and triangular inequality (TR) $(d(x, z) \leq d(x, y) + d(y, z))$. It turns out that $\mathcal{CSL}$ cannot distinguish between minspace models which satisfy (TR) from models which do not. In contrast ([9]), $\mathcal{CSL}$ has enough expressive power in order to distinguish between symmetric and non-symmetric minspace models. As a first step, we concentrate here on the general non-symmetric case, leaving the interesting symmetric case to further research.

$\mathcal{CSL}$ is a logic of pure qualitative comparisons. This motivates an alternative semantics where the distance function is replaced by a family of comparisons relations, one for each object. We call this semantics *preferential* semantics, similarly to the semantics of conditional logics [8,7]. Preferential structures are equipped by a family of strict pre-orders. We may interpret this relations as expressing a comparative similarity between objects. For three objects, $x \prec_w y$ states that w is more similar to x than to y.

The preferential semantics in itself is more general than distance model semantics. However, if we assume the additional conditions of the definition 2, it turns out that these two are equivalent (theorem 4).

Definition 2. *We will say that a preferential relation $\prec_w$ over Δ:*

(i) is modular *iff $\forall x, y, z \in \Delta$, $(x \prec_w y) \to (z \prec_w y \ \vee \ x \prec_w z)$.*
(ii) is centered *iff $\forall x \in \Delta$, $x = w \ \vee \ w \prec_w x$.*
(iii) satisfies the Limit Assumption *iff $\forall X \subseteq \Delta$, $X \neq \emptyset \ \to \ \min_{\prec_w}(X) \neq \emptyset$,*[1]
where $\min_{\prec_w}(X) = \{y \in X \mid \forall z(z \prec_w y \to z \notin X)\}$.

[1] We note that the Limit Assumption implies that the preferential relation is asymmetric. On the other hand, on a finite set, asymmetry implies Limit Assumption. Modularity and asymmetry imply that the relation is also transitive and irreflexive.

Modularity is strongly related to the fact that the preferential relations represent distance comparisons. This is the key property to enforce the equivalence with distance models. Centering states that w is the *unique* minimal element for its preferential relation $\prec_w$, and can be seen as the preferential counterpart of (ID). The Limit Assumption states that each non-empty set has at least one minimal element with respect to any preferential relation (i.e it does not contain an infinitely descending chain), and corresponds to (MIN).

Definition 3 ($\mathcal{CSL}$-preferential model). *A $\mathcal{CSL}$-preferential model is a triple $\mathcal{M} = (\Delta, (\prec_w)_{w\in\Delta}, .^{\mathcal{M}})$ where:*

- *Δ is a non-empty set of* objects *(or* possible worlds*).*
- *$(\prec_w)_{w\in\Delta}$ is a family of* preferential relation*, each one being modular, centered, and satisfying the* limit assumption*.*
- *$.^{\mathcal{M}}$ is the evaluation function defined as in definition 1, except for $\leftleftarrows$:*

$$(A \leftleftarrows B)^{\mathcal{M}} = \left\{ w \in \Delta \middle| \exists x \in A^{\mathcal{M}} \text{ such that } \forall y \in B^{\mathcal{M}},\ x \prec_w y \right\} .$$

Validity is defined as in definition 1.

We now show the equivalence between preferential models and distance minspace models. We say that a $\mathcal{CSL}$-preferential model $\mathcal{I}$ and a $\mathcal{CSL}$-distance minspace model $\mathcal{J}$ are *equivalent* iff they are based on the same set Δ, and for all formulas $A \in \mathcal{L}_{\mathcal{CSL}}$, $A^{\mathcal{I}} = A^{\mathcal{J}}$.

Theorem 4 (Equivalence between $\mathcal{CSL}$-preferential models and $\mathcal{CSL}$-distance models)

1. *For each $\mathcal{CSL}$-distance minspace model, there is an equivalent $\mathcal{CSL}$-preferential model.*
2. *For each $\mathcal{CSL}$-preferential model, there is an equivalent $\mathcal{CSL}$-distance minspace model.*

Proof 1. ([9]): given $\mathcal{I} = (\Delta^{\mathcal{I}}, d, .^{\mathcal{I}})$ a $\mathcal{CSL}$-distance minspace model, just define a preferential model $\mathcal{J}$ by stipulating $x \prec_w y$ iff $d(w,x) < d(w,y)$, and for all propositional variables V_i, $V_i^{\mathcal{J}} = V_i^{\mathcal{I}}$. It is easy to check that $\prec_w$ is modular, centered, and satisfies the limit assumption, and that $\mathcal{I}$ and $\mathcal{J}$ are equivalent.

2. Since the relation $\prec_w$ is modular, we can assume that there exists a *ranking function*[2] $r_w : \Delta \to \mathbb{R}^{\geq 0}$ such that $x \prec_w y$ iff $r_w(x) < r_w(y)$. Therefore, given a $\mathcal{CSL}$-preferential model $\mathcal{J} = (\Delta^{\mathcal{J}}, (\prec_w)_{w\in\Delta^{\mathcal{J}}}, .^{\mathcal{J}})$, we can define a $\mathcal{CSL}$-distance minspace model $\mathcal{I} = (\Delta^{\mathcal{J}}, d, .^{\mathcal{J}})$, where the distance function d is defined as follow: if $w = x$ then $d(w,x) = 0$, and $d(w,x) = r_w(x)$ otherwise. We can easily check that (i) $\mathcal{I}$ is a minspace because of the limit assumption, and that (ii) $\mathcal{I}$ and $\mathcal{J}$ are equivalent; this is proved by induction on the complexity of formulas.

[2] See for instance [6], Lemma 14.

(1) $\neg(A \leftleftarrows B) \sqcup \neg(B \leftleftarrows A)$ (2) $(A \leftleftarrows B) \rightarrow (A \leftleftarrows C) \sqcup (C \leftleftarrows B)$

(3) $A \sqcap \neg B \rightarrow (A \leftleftarrows B)$ (4) $(A \leftleftarrows B) \rightarrow \neg B$

(5) $(A \leftleftarrows B) \sqcap (A \leftleftarrows C) \rightarrow (A \leftleftarrows (B \sqcup C))$ (6) $(A \leftleftarrows \bot) \rightarrow \neg(\neg(A \leftleftarrows \bot) \leftleftarrows \bot)$

$(Mon)\ \dfrac{\vdash (A \rightarrow B)}{\vdash (A \leftleftarrows C) \rightarrow (B \leftleftarrows C)}$ $(Taut)$ Classical tautologies and rules.

Fig. 1. CSMS axioms

We have mentioned the relation with conditional logics. These logics, originally introduced by Lewis and Stalnaker [7,12], contain a connective $A > B$ whose reading is approximatively "if A were true then B would also be true"[3]. The idea is that a world/state x verifies $A > B$ if B holds in all states y that are most similar to x that is:

$$x \in (A > B)^{\mathcal{M}} \text{ iff } \min_{\prec_x}(A^{\mathcal{M}}) \subseteq B^{\mathcal{M}} \ .$$

The two connectives $\leftleftarrows$ are interdefinable as shown in [9]:

$$A > B \equiv (A \leftleftarrows (A \wedge \neg B)) \vee \neg(A \leftleftarrows \bot) \ ,$$

$$A \leftleftarrows B \equiv ((A \vee B) > A) \wedge (A > \neg B) \wedge \neg(A > \bot) \ .$$

By means of this equivalence, an (indirect) axiomatization of $\leftleftarrows$ can be obtained: just take an axiomatization of the suitable conditional logic (well known) and add the definition above. On the other hand an axiomatisation of $\mathcal{CSL}$ over arbitrary distance models is presented in [11], however it makes use of an extended language, as we comment below. Moreover, the case of minspaces has not been studied in details. Our axiomatisation is contained in fig. 1. The axioms (1) and (2) capture respectively the asymmetry and modularity of the preference relations, whereas (3) and (4) are enforced by centering and the minspace property[4]. By (5), we obtain that $\leftleftarrows$ distributes over disjunction on the second argument, since the opposite direction is derivable. The axiom (6) is similar to axiom (33) of the axiomatization in [11], it says that the modality $\Diamond A \equiv A \leftleftarrows \bot$ has the properties of **S5**. Finally, the rule (Mon) states the monotonicity of $\leftleftarrows$ in the first argument, a dual rule stating the anti-monotonicity in the second argument is derivable as well.

The axiomatisation of $\mathcal{CSL}$ provided in [11] for arbitrary distance spaces makes use of the operator $\circ_R A$ that, referring to preferential models, selects

[3] To this regard, in alternative to the concept/subset interpretation mentioned so far, the formula $A \leftleftarrows B$ may perhaps be read as "A is (strictly) more plausible than B". This interpretation may intuitively explain the relation with the conditional operator.

[4] Observe that the minspace property (or Limit Assumption as it is called in condtional logics.) in itself is not definable by any formula of $\mathcal{CSL}$.

elements x for which $\min_{\prec_x}(A)$ is non-empty. As observed in [11], an axiomatization of $\mathcal{CSL}$ over minspaces can then be obtained by just adding the axiom $\circ_R A \leftrightarrow (A \leftleftarrows \bot)$. However our axiomatization is significantly simpler (almost one half of the axioms). We can show that our axiomatization is sound and complete with respect to the preferential semantics, whence with respect to minspace models (by theorem 4).

Theorem 5. *A formula is derivable in* **CSMS** *iff it is valid in every $\mathcal{CSL}$-preferential model.*[5]

3 A Tableaux Calculus

In this section, we present a tableau calculus for $\mathcal{CSL}$, this calculus provides a decision procedure for this logic. We identify a tableau with a set of sets of formulas $\Gamma_1, \ldots, \Gamma_n$. Each Γ_i is called a *tableau set*[6]. Our calculus will make use of labels to represent objects of the domain. Let us consider formulas $(A \leftleftarrows B)$ and $\neg(A \leftleftarrows B)$ under preferential semantics. We have:

$$w \in (A \leftleftarrows B)^{\mathcal{M}} \text{ iff } \exists x(x \in A^{\mathcal{M}} \wedge \forall z(z \in B^{\mathcal{M}} \rightarrow x \prec_w z)) \ .$$

In minspace models, the right part is equivalent to:

$$w \in (A \leftleftarrows B)^{\mathcal{M}} \text{ iff } \exists u \in A^{\mathcal{M}} \text{ and } \forall y(y \in B^{\mathcal{M}} \rightarrow \exists x(x \in A^{\mathcal{M}} \wedge x \prec_w y)) \ .$$

We now introduce a pseudo-modality $\Box_w$ indexed on objects:

$$x \in (\Box_w A)^{\mathcal{M}} \text{ iff } \forall y(y \prec_w x \rightarrow y \in A^{\mathcal{M}}) \ .$$

Its meaning is that $x \in (\Box_w A)^{\mathcal{M}}$ iff A holds in all worlds preferred to x with respect to $\prec_w$. Observe that we have then the equivalence:

Claim 1. $w \in (A \leftleftarrows B)^{\mathcal{M}}$ *iff* $A^{\mathcal{M}} \neq \emptyset$ *and* $\forall y(y \notin B^{\mathcal{M}}$ *or* $y \in (\neg\Box_w\neg A)^{\mathcal{M}})$.

This equivalence will be used to decompose $\leftleftarrows$-formulas in an analytic way. The tableau rules make also use of a universal modality $\Box$ (and its negation). The language of tableaux comprises the following kind of formulas: $x : A, x : (\neg)\Box\neg A, x : (\neg)\Box_y\neg A, x <_y z$, where x, y, z are labels and A is a $\mathcal{CSL}$-formula. The meaning of $x : A$ is the obvious one: $x \in A^{\mathcal{M}}$. The reading of the rules is the following: we apply a rule

$$\frac{\Gamma[E_1, \ldots, E_k]}{\Gamma_1 \mid \ \ldots \ \mid \Gamma_n}$$

to a tableau set Γ if each formula E_i $(1 \leq i \leq k)$ is in Γ. We then replace Γ with any tableau set $\Gamma_1, \ldots, \Gamma_n$. As usual, we let Γ, A stand for for $\Gamma \cup \{A\}$, where A is a tableau formula. The tableaux rules are shown in figure Figure 2.

[5] The full proofs of all results reported in this paper are contained in [1].

[6] A tableau set corresponds to a *branch* in a tableau-as-tree representation.

$$(T\sqcap)\quad \frac{\Gamma[x : A \sqcap B]}{\Gamma,\ x : A,\ x : B} \qquad\qquad (F\sqcap)\quad \frac{\Gamma[x : \neg(A \sqcap B)]}{\Gamma,\ x : \neg A \mid \Gamma,\ x : \neg B}$$

$$(NEG)\quad \frac{\Gamma[x : \neg\neg A]}{\Gamma, x : A} \qquad\qquad (F1 \leftleftarrows)\quad \frac{\Gamma[x : \neg(A \leftleftarrows B)]}{\Gamma, x : \Box\neg A \mid \Gamma, x : B \mid \Gamma, x : \neg A,\ x : \neg B}$$

$$(T \leftleftarrows)(*)\quad \frac{\Gamma[x : A \leftleftarrows B]}{\Gamma, x : \neg\Box\neg A,\ y : \neg B \mid \Gamma, y : B, y : \neg\Box_x\neg A} \qquad\qquad (F2 \leftleftarrows)(**)\quad \frac{\Gamma[x : \neg(A \leftleftarrows B),\ x : \neg A,\ x : \neg B]}{\Gamma,\ y : B,\ y : \Box_x\neg A}$$

$$(F1\Box_x)\quad \frac{\Gamma[z : \neg\Box_x\neg A]}{\Gamma, x : \neg A \mid \Gamma, x : A}$$

$$(T\Box_x)(*)\quad \frac{\Gamma[z : \Box_x\neg A, y <_x z]}{\Gamma,\ y : \neg A,\ y : \Box_x\neg A} \qquad\qquad (F2\Box_x)(**)\quad \frac{\Gamma[z : \neg\Box_x\neg A,\ x : \neg A]}{\Gamma,\ y <_x z,\ y : A,\ y : \Box_x\neg A}$$

$$(T\Box)(*)\quad \frac{\Gamma[x : \Box\neg A]}{\Gamma,\ y : \neg A, y : \Box\neg A} \qquad\qquad (F\Box)(**)\quad \frac{\Gamma[x : \neg\Box\neg A]}{\Gamma,\ y : A}$$

$$(Mod)(*)\quad \frac{\Gamma[z <_x u]}{\Gamma, z <_x y \mid \Gamma, y <_x u} \qquad\qquad (Cent)(***)\quad \frac{\Gamma}{\Gamma, x <_x y \mid \Gamma[x/y]}$$

(*) y is a label occurring in Γ. (**) y is a new label not occurring in Γ. (***) x and y are two distinct labels occurring in Γ.

Fig. 2. Tableau rules for $\mathcal{CSL}$

Let us comment on the rules which are not immediately obvious. The rule for $(T \leftleftarrows)$ encodes directly the semantics by virtue of claim 1. However in the negative case the rule is split in two: if x satisfies $\neg(A \leftleftarrows B)$, either A is empty, or there must be an $y \in B$ such that there is no $z \prec_x y$ satisfying A; if x satisfies B then x itself fulfills this condition, i.e. we could take $y = x$, since x is $\prec_x$-minimal (by centering). On the other hand, if x does not satisfies B, then x cannot satisfy A either (otherwise x would satisfy $A \leftleftarrows B$) and there must be an y as described above. This case analysis with respect to x is performed by the (F1$\leftleftarrows$) rule, whereas the creation y for the latter case is performed by $(F2 \leftleftarrows)$. We have a similar situation for the (F$\Box_x$) rule: let z satisfy $\neg\Box_x\neg A$, then there must be an $y \prec_x z$ satisfying A; but if x satisfies A we can take $x = y$, since $x \prec_x z$ (by centering). If x does not satisfy A then we must create a suitable y and this is the task of the (F2$\Box_x$) rule. Observe that the rule does not simply create a $y \prec_x z$ satisfying A but it creates a *minimal* one. The rule is similar to the (F$\Box$) rule in modal logic GL (Gödel-Löb modal logic of arithmetic provability) [2] and it is enforced by the Limit Assumption. This formulation of the rules for (F$\leftleftarrows$) and for (F$\Box_x$) prevents the unnecessary creation of new objects whenever the existence of the objects required by the rules is assured by centering. The rule $(Cent)$ is of a special kind: it has no premises (ie. it can always be applied) and generates two tableau sets: one with $\Gamma \cup \{x <_x y\}$, where

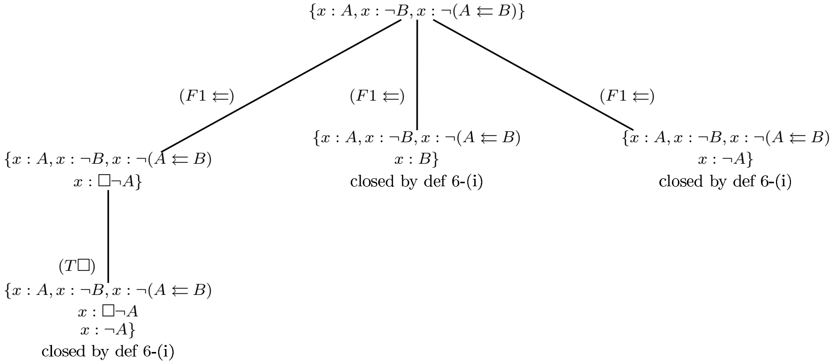

$\{x : A, x : \neg B, x : \neg(A \leftleftarrows B)\}$
$(F1 \leftleftarrows)$
$(F1 \leftleftarrows)$
$(F1 \leftleftarrows)$
$\{x : A, x : \neg B, x : \neg(A \leftleftarrows B)$
$x : \Box\neg A\}$
$\{x : A, x : \neg B, x : \neg(A \leftleftarrows B)$
$x : B\}$
closed by def 6-(i)
$\{x : A, x : \neg B, x : \neg(A \leftleftarrows B)$
$x : \neg A\}$
closed by def 6-(i)
$(T\Box)$
$\{x : A, x : \neg B, x : \neg(A \leftleftarrows B)$
$x : \Box\neg A$
$x : \neg A\}$
closed by def 6-(i)

Fig. 3. An example of tableau: provability of $A \sqcap \neg B \rightarrow (A \leftleftarrows B)$

x and y are two distinct labels occurring in Γ), and one where we replace x by y in Γ, i.e. where we identify the two labels.

Definition 6 (Closed set, closed tableau). *A tableau set Γ is* closed *if one of the three following conditions hold: (i) $x : A \in \Gamma$ and $x : \neg A \in \Gamma$, for any formula A, or $x : \bot \in \Gamma$. (ii) $y <_x z$ and $z <_x y$ are in Γ. (iii) $x : \neg\Box_x A \in \Gamma$.*

A $\mathcal{CSL}$-tableau is closed if every tableau set is closed.

In order to prove soundness and completeness of the tableaux rules, we introduce the notion of satisfiability of a tableau set by a model.

Given a tableau set Γ, we denote by Lab_Γ the set of labels occurring in Γ.

Definition 7 ($\mathcal{CSL}$-mapping, satisfiable tableau set)
Let $\mathcal{M} = (\Delta, (\prec_w)_{w \in \Delta}, .^{\mathcal{M}})$ be a preferential model, and Γ a tableau set. A $\mathcal{CSL}$-mapping from Γ to $\mathcal{M}$ is a function $f : \mathrm{Lab}_\Gamma \longrightarrow \Delta$ satisfying the following condition: for every $y <_x z \in \Gamma$, we have $f(y) \prec_{f(x)} f(z)$ in $\mathcal{M}$.

We say that Γ is satisfiable under f in $\mathcal{M}$ if $x : A \in \Gamma$ implies $f(x) \in A^{\mathcal{M}}$. A tableau set Γ is satisfiable if it is satisfiable in some $\mathcal{CSL}$-preferential model $\mathcal{M}$ under some $\mathcal{CSL}$-mapping f. A $\mathcal{CSL}$-tableau is satisfiable if at least one of its sets is satisfiable.

We can show that our tableau calculus is sound and complete with respect to the preferential semantics, whence with respect to minspace models (by theorem 4).

Theorem 8 (Soundness of the calculus). *If a formula $A \in \mathcal{L}_{\mathcal{CSL}}$ is satisfiable with respect to preferential semantics then any tableau begining with $\{x : A\}$ is open.*

The proof of the soundness is standard, and can be found in [1]. In order to show completeness, we need the following definition:

Definition 9 (Saturated tableau set). *We say that a tableau set Γ is saturated iff it satisfies the following conditions:*

($T\sqcap$) *If* $x : A \sqcap B \in \Gamma$ *then* $x : A \in \Gamma$ *and* $x : B \in \Gamma$.
($F\sqcap$) *If* $x : \neg(A \sqcup B) \in \Gamma$ *then* $x : \neg A \in \Gamma$ *or* $x : \neg B \in \Gamma$.
(NEG) *If* $x : \neg\neg A \in \Gamma$ *then* $x : A \in \Gamma$.
($T \leftleftarrows$) *If* $x : (A \leftleftarrows B) \in \Gamma$ *then for all* $y \in \mathrm{Lab}_\Gamma$, *either* $y : \neg B \in \Gamma$ *and* $\neg\Box\neg A \in \Gamma$, *or* $y : B$ *and* $y : \neg\Box_x\neg A$ *are in* Γ.
($F \leftleftarrows$) *If* $x : \neg(A \leftleftarrows B) \in \Gamma$ *then either (i)* $x : \Box\neg A \in \Gamma$, *or (ii)* $x : B \in \Gamma$, *or (iii)* $x : \neg A$ *and* $x : \neg B$ *are in* Γ *and there exists* $y \in \mathrm{Lab}_\Gamma$ *such that* $y : B$ *and* $y : \Box_x\neg A$ *are in* Γ.
($T\Box_x$) *If* $z : \Box_x\neg A \in \Gamma$ *and* $y <_x z \in \Gamma$, *then* $y : \neg A$ *and* $y : \Box_x\neg A$ *are in* Γ.
($F\Box_x$) *If* $z : \neg\Box_x\neg A \in \Gamma$, *then either (i)* $x : A \in \Gamma$, *or (ii)* $x : \neg A \in \Gamma$ *and there exists* $y \in \mathrm{Lab}_\Gamma$ *such that* $y <_x z$, $y : A$ *and* $y : \Box_x\neg A$ *are in* Γ.
($T\Box$) *If* $x : \Box\neg A \in \Gamma$, *then for all* $y \in \mathrm{Lab}_\Gamma$, $y : \neg A$ *and* $y : \Box\neg A$ *are in* Γ.
($F\Box$) *If* $x : \neg\Box\neg A \in \Gamma$, *then there is* $y \in \mathrm{Lab}_\Gamma$ *such that* $y : A \in \Gamma$.
($Cent$) *For all* $x, y \in \mathrm{Lab}_\Gamma$ *such that* $x \neq y$, $x <_x y$ *is* $\in \Gamma$.
(Mod) *If* $y <_x z \in \Gamma$, *then for all labels* $u \in \mathrm{Lab}_\Gamma$, *either* $u <_x z \in \Gamma$, *or* $y <_x u \in \Gamma$.

We say that Γ *is saturated with respect to a rule* R *iff* Γ *satisfies the corresponding saturation condition for* R *of the above definition.*

The following lemma shows that the preference relations $<_x$ satisfies the Limit Assumption for an open tableau set.

Lemma 10 (See [1]). *Let* Γ *be an open tableau set containing only a finite number of positive* $\leftleftarrows$*-formulas* $x : A_0 \leftleftarrows B_0$, $x : A_1 \leftleftarrows B_1$, $x : A_2 \leftleftarrows B_2$, $\ldots$, $x : A_{n-1} \leftleftarrows B_{n-1}$. *Then* Γ *does not contain any infinite descending chain of labels* $y_1 <_x y_0$, $y_2 <_x y_1$, $\ldots$, $y_{i+1} <_x y_i, \ldots$.

Theorem 11. *If* Γ *is an open and saturated tableau set, then* Γ *is satisfiable.*

(Proof): Given an open tableau set Γ, we define a canonical model $\mathcal{M}_\Gamma = \langle \Delta, (\prec_w)_{w\in\Delta}, .^{\mathcal{M}_\Gamma} \rangle$ as follows:

- $\Delta = \mathrm{Lab}_\Gamma$ and $y \prec_x z$ iff $y <_x z \in \Gamma$.
- For all propositional variables $V_i \in \mathcal{V}_p$, $V_i^{\mathcal{M}_\Gamma} = \{x \mid x : V_i \in \Gamma\}$.

$\mathcal{M}_\Gamma$ is indeed a $\mathcal{CSL}$-model, as each preferential relation is centered, modular, and satisfies the limit assumption. The first two came from the rules ($Cent$) and (Mod), and we have the latter by lemma 10.

We now show that Γ is satisfiable in $\mathcal{M}_\Gamma$ under the trivial identity mapping, i.e for all formula $C \in \mathcal{L}_{\mathcal{CSL}}$: (i) if $x : C \in \Gamma$, then $x \in C^{\mathcal{M}_\Gamma}$. (ii) if $x : \neg C \in \Gamma$, then $x \in (\neg C)^{\mathcal{M}_\Gamma}$.

Proof. We reason by induction on the complexity $cp(C)$ of a formula C, where we suppose that $cp((\neg)\Box\neg A), cp((\neg)\Box_x\neg A) < cp(A \leftleftarrows B)$.

- if $C = V_i, C \in \mathcal{V}_p$, $x \in C^{\mathcal{M}_\Gamma}$ by the definition of $\mathcal{M}_\Gamma$.
- For the boolean formulas, the proof is standard.

- if $C = A \leftleftarrows B$: since Γ is saturated, for every $y \in \text{Lab}_\Gamma$ we have either $x : \neg\Box\neg A \in \Gamma$ and $y : \neg B \in \Gamma$, or $y : B \in \Gamma$ and $y : \neg\Box_x\neg A \in \Gamma$. By induction hypothesis, in the first case we get $A^{\mathcal{M}_\Gamma} \neq \emptyset$ and $y \notin B^{\mathcal{M}_\Gamma}$, and in the second we have that $y \in (\neg\Box_x\neg A)^{\mathcal{M}_\Gamma}$ which also entails $A^{\mathcal{M}_\Gamma} \neq \emptyset$. Thus, by claim 1, we have $x \in (A \leftleftarrows B)^{\mathcal{M}_\Gamma}$.
- if $C = \neg(A \leftleftarrows B)$: by the saturation conditions, we have 3 cases.
  (a) $x : \Box\neg A \in \Gamma$. By application of the rule $(T\Box)$, for all label y, $y : \neg A \in \Gamma$. By our induction hypothesis, $A^{\mathcal{M}_\Gamma} = \emptyset$, and so $x \in (\neg(A \leftleftarrows B))^{\mathcal{M}_\Gamma}$.
  (b) $x : B \in \Gamma$. By induction hypothesis, $x \in B^{\mathcal{M}_\Gamma}$, and so $x \in (\neg(A \leftleftarrows B))^{\mathcal{M}_\Gamma}$ by axiom (4).
  (c) $x : \neg A, x : \neg B$ are in Γ, and there is a label y such that $y : B, y : \Box_x\neg A$ are in Γ. By induction hypothesis, we have $y \in B^{\mathcal{M}_\Gamma}$ and $y \in (\Box_x\neg A)^{\mathcal{M}_\Gamma}$, so that by claim 1, we have $x \in (\neg(A \leftleftarrows B))^{\mathcal{M}_\Gamma}$.
- if $C = \Box_y\neg A$, by saturation we have: for all z, if $z <_x \in \Gamma$ then $z : \neg A \in \Gamma$ and $z : \Box_y\neg A \in \Gamma$. Then by induction hypothesis, we have that for all z, if $z \prec_y x$ then $z \in (\neg A)^{\mathcal{M}_\Gamma}$ which means that $x \in (\Box_y\neg A)^{\mathcal{M}_\Gamma}$.
- if $C = \neg\Box_y\neg A$, by saturation we have either $y : A \in \Gamma$ or $y : \neg A \in \Gamma$. In the first case, since $y \neq x$ (or Γ would be closed by def 6-(iii)) and $\prec_y$ satisfies centering we have $y \prec_y x$, and by induction hypothesis, $y \in A^{\mathcal{M}_\Gamma}$. Thus $x \in (\neg\Box_y\neg A)^{\mathcal{M}_\Gamma}$. In the second case, by saturation there is $z \in \text{Lab}_\Gamma$ such that $z <_y x \in \Gamma$, $z : A \in \Gamma$. By induction hypothesis and the definition of $\prec_y$, we conclude that $x \in (\neg\Box_y\neg A)^{\mathcal{M}_\Gamma}$.

4 Termination of the Tableau Calculus

The calculus presented above can lead to non-terminating computations due to the interplay between the rules which generate new labels (the dynamic rules $(F2 \leftleftarrows)$, $(F\Box)$ and $(F\Box_x)$) and the static rule $(T \leftleftarrows)$ which generates formula $\neg\Box_x A$ to which $(F\Box_x)$ may again be applied. Our calculus can be made terminating by defining a systematic procedure for applying the rules and by introducing appropriate blocking conditions. The systematic procedure simply prescribes to apply static rules as far as possible before applying dynamic rules. To prevent the generation of an infinite tableau set, we put some restrictions on the rule's applications. The restrictions on all rules except $(F2 \leftleftarrows)$ and $(F2\Box_x)$ are easy and prevent redundant applications of the rules. We call the restrictions on $(F2 \leftleftarrows)$ and $(F2\Box_x)$ *blocking conditions* in analogy with standard conditions for getting termination in modal and description logics tableaux; they prevent the generation of infinitely many labels by performing a kind of loop-checking.

To this aim, we first define a total ordering $\sqsubset$ on the labels of a tableau set such that $x \sqsubset y$ for all labels x that are already in the tableau when y is introduced. If $x \sqsubset y$, we will say that x is older than y.

We define $\text{Box}^+_{\Gamma,x,y}$ as the set of positive boxed formulas indexed by x labelled by y which are in Γ: $\text{Box}^+_{\Gamma,x,y} = \{\Box_x\neg A \mid y : \Box_x\neg A \in \Gamma\}$ and $\Pi_\Gamma(x)$ as the set of non boxed formulas labelled by x: $\Pi_\Gamma(x) = \{A \mid A \in \mathcal{L}_{\mathcal{CSL}}$ and $x : A \in \Gamma\}$.

Definition 12
(Static and dynamic rules) *We call* dynamic *the following rules:* $(F2 \leftleftarrows)$*,* $(F2\Box_x)$ *and* $(F\Box)$*. We call* static *all the other rules.*
(Rules restrictions)

1. *Do not apply a static rule to* Γ *if at least one of the consequences is already in it.*
2. *Do not apply the rule* $(F2 \leftleftarrows)$ *to a* $x : \neg(A \leftleftarrows B), x : \neg A, x : \neg B$
   (a) *if there exists some label* y *in* Γ *such that* $y : B$ *and* $y : \Box_x \neg A$ *are in* Γ*.*
   (b) *if there exists some label* u *such that* $u \sqsubset x$ *and* $\Pi_\Gamma(x) \subseteq \Pi_\Gamma(u)$*.*
3. *Do not apply the rule* $(F2\Box_x)$ *to a* $z : \neg\Box_x\neg A, x : \neg A$
   (a) *if there exists some label* y *in* Γ *such that* $y <_x z$*,* $y : A$ *and* $y : \Box_x\neg A$ *are in* Γ*.*
   (b) *if there exists some label* u *in* Γ *such that* $u \sqsubset x$ *and* $\Pi_\Gamma(x) \subseteq \Pi_\Gamma(u)$*.*
   (c) *if there exists some label* v *in* Γ *such that* $v \sqsubset z$ *and* $v : \neg\Box_x\neg A \in \Gamma$ *and* $\mathrm{Box}^+_{\Gamma,x,z} \subseteq \mathrm{Box}^+_{\Gamma,x,v}$*.*
4. *Do not apply the rule* $(F\Box)$ *to a* $x : \neg\Box\neg A$ *in* Γ *if there exists some label* y *such that* $y : A$ *is in* Γ*.*

(Systematic procedure) *(1) Apply static rules as far as possible. (2) Apply a (non blocked) dynamic rule to some formula labelled* x *only if no dynamic rule is applicable to a formula labelled* y*, such that* $y \sqsubset x$*.*

We prove that a tableau initialized with a $\mathcal{CSL}$-formula always terminates provided it is expanded according to Definition 12.

Theorem 13. *Let* Γ *be obtained from* $\{x : A\}$*, where* A *is a* $\mathcal{CSL}$*-formula, by applying an arbitrary sequence of rules respecting definition 12. Then* Γ *is finite.*

Proof. Suppose by absurdity that Γ is not finite. Since the static rules (and also the $(F\Box)$ rule) may only add a finite of number of formulas for each label, Γ must contain an infinite number of labels generated by the dynamic rules, either $(F2 \leftleftarrows)$ or $(F2\Box_x)$ (or both).

Let Γ contain infinitely many labels introduced by $(F2 \leftleftarrows)$. Since the number of negative $\leftleftarrows$ formulas is finite, there must be one formula, say $\neg(B \leftleftarrows C)$, such that for an infinite sequence of labels $x_1, \ldots, x_i, \ldots$, $x_i : \neg(B \leftleftarrows C) \in \Gamma$. By blocking condition (2b) we then have that for every i, $\Pi_\Gamma(x_i) \not\subseteq \Pi_\Gamma(x_1), \ldots, \Pi_\Gamma(x_i) \not\subseteq \Pi_\Gamma(x_{i-1})$. But this is impossible since each $\Pi_\Gamma(x_i)$ is finite (namely bounded by $O(|A|)$) and the rules are non-decreasing with respect to $\Pi_\Gamma(x_i)$ (an application of a rule can never remove formulas from $\Pi_\Gamma(x_i)$).

Let now Γ contain infinitely many labels introduced by $(F2\Box_x)$. That is to say, Γ contains $x_i : \neg\Box_{y_i}\neg B$ for infinitely many x_i and y_i. If all y_i are distinct, Γ must contain in particular infinitely many formulas $x : \neg\Box_{y_i}\neg B$ for a fixed x. The reason is that $x_i : \neg\Box_{y_i}\neg B$ may only be introduced by applying $(T \leftleftarrows)$, thus there must be infinitely many $y_i : B \leftleftarrows C \in \Gamma$. By the systematic procedure, the rule (T $\leftleftarrows$) has been applied to a label x for every $y_i : B \leftleftarrows C \in \Gamma$ generating $x : \neg\Box_{y_i}\neg B$ for all y_i. But then we can find a contradiction with respect to blocking condition (3b) as in the previous case, since for each i we would have $\Pi_\Gamma(y_i) \not\subseteq \Pi_\Gamma(y_1), \ldots, \Pi_\Gamma(y_i) \not\subseteq \Pi_\Gamma(y_{i-1})$. We can conclude that Γ cannot contain $x_i : \neg\Box_{y_i}\neg B$, for infinitely many distinct y_i and distinct x_i. We are left with the case Γ contains $x_i : \neg\Box_y\neg B$ for a fixed y and infinitely many x_i. In this case, by blocking condition (3c), we have that for each i, $\mathrm{Box}^+_{\Gamma,y,x_i} \not\subseteq \mathrm{Box}^+_{\Gamma,y,x_1}, \ldots, \mathrm{Box}^+_{\Gamma,y,x_i} \not\subseteq \mathrm{Box}^+_{\Gamma,x_{i-1}}$. But again this is impossible given the fact that each $\mathrm{Box}^+_{\Gamma,y,x_i}$ is finite (bounded by $O(|A|)$) and that the rules are non-decreasing with respect to the sets Box^+_Γ.

To prove completeness, we will consider tableau sets saturated under blocking. A tableau set Γ is saturated under blocking iff (a) it is build according to Definition 12 (b) No further rules can be applied to it. It is easy to see that if Γ is saturated under blocking, it satisfies all the saturation conditions in Definition 9 except possibly for conditions $(F \leftleftarrows)$.(iii) and $(F\Box_x)$.(ii).

By the termination theorem, we get that any tableau set generated from an initial set containing just a $\mathcal{CSL}$ formula, will be either closed or saturated under blocking in a finite number of steps.

We now show that an open tableau set saturated under blocking can be extended to an open saturated tableau set, that is satisfying all conditions of definition 9. By means of theorem 8 we obtain the completeness of the terminating procedure.

Theorem 14. *If Γ is saturated and open under blocking, then there exists an open and saturated set Γ^* such that $\forall A \in \mathcal{L}_{\mathcal{CSL}}$, if $x : A \in \Gamma$ then $x : A \in \Gamma^*$.*

Let Γ be an open and saturated set under blocking. We will construct the set Γ^* from Γ in three steps. First, we consider formulas $z : \neg\Box_x\neg A$ which are blocked by condition 3c (and not by 3b). We construct a set Γ_1 from Γ which satisfies the saturation condition $(F\Box_x)$ with respect to these formulas.

Step 1. For each formula $z : \neg\Box_x\neg A \in \Gamma$ for which condition $(F\Box_x)$ is not fulfilled and that is blocked only by condition 3c, we consider the oldest label u that blocks the formula. Therefore, the formula $u : \neg\Box_x\neg A$ is in Γ and it is not blocked by condition 3c [7]. Since $z : \neg\Box_x\neg A$ is not blocked by condition 3b, $u : \neg\Box_x\neg A$ is not blocked for this condition either, and thus the rule $(F2\Box_x)$ has been applied to it. Hence there exists a label y such that $y : A, y : \Box_x\neg A$ and $y <_x u$ are in Γ. We then add $y <_x z$ to Γ. We call Γ_1 the resulting set.

Claim 2. *(I) Γ_1 is saturated, except for (Mod) and the formulas $x : \neg(A \leftleftarrows B)$ and $z : \neg\Box_x\neg A$ respectively blocked by condition 2b and 3b. (II) It is open.*

The step 2 will now build a set Γ_2 saturated with respect to (Mod) from Γ_1.

Step 2. For each $y <_x z \in \Gamma$, if $\mathrm{Box}^+_{\Gamma,x,z} \subset \mathrm{Box}^+_{\Gamma,x,y}$, then for each z_0 such that $\mathrm{Box}^+_{\Gamma,x,z_0} = \mathrm{Box}^+_{\Gamma,x,z}$ we add $y <_x z_0$ to Γ_1. We call Γ_2 the resulting set.

[7] If it was, let v older than u the label which causes the blocking. Then v will also block $u : \neg\Box_x\neg A$, contradiction, as u is by hypothesis the oldest label blocking $z : \neg\Box_x\neg A$.

Claim 3. *(I) Γ_2 is saturated except for the formulas $x : \neg(A \leftleftarrows B)$ and $z : \neg\Box_x\neg A$ respectively blocked by condition 2b and 3b. (II) It is open.*

We will now consider the formulas blocked by conditions 2b and 3b, and finally build a set Γ_3 saturated with respect to all rules from Γ_2.

Step 3. For each label x such that there is a formula $x : \neg(A \leftleftarrows B) \in \Gamma$ or $z : \neg\Box_x\neg A \in \Gamma$ respectively blocked by condition 2b or 3b, we let u be the oldest label which caused the blocking. We then construct the set Γ_3 by the following procedure:

1. we remove from Γ_2 each relation $<_x$, and all formulas $v : \neg\Box_x\neg A$ and $v : \Box_x\neg A$ ($v \in \mathrm{Lab}_\Gamma$).
2. For all label $z \in \mathrm{Lab}_\Gamma$ such that $z \neq x$, we add $x <_x z$.
3. For all labels $z, v \in \mathrm{Lab}_\Gamma$ such that $z \neq x$, if $v <_u z \in \Gamma_2$, then we add $v <_x z$.
4. For each $v : \Box_u\neg A \in \Gamma$, if $A \in \Pi_\Gamma(x)$ we then add $v : \Box_x\neg A$.
5. For each $v : \neg\Box_u\neg A \in \Gamma$ such that $v \neq x$, we add $v : \neg\Box_u\neg A$.
6. For each formula $A \in \Pi_\Gamma(x)$, we add $x : \Box_x A$.

Claim 4. *(I) Γ_3 is saturated with respect to all rules. (II) It is open.*

We then let $\Gamma^* = \Gamma_3$. It is easy to see that for all formulas $A \in \mathcal{L}_{\mathcal{CSL}}$, if $x : A \in \Gamma$ then $x : A \in \Gamma^*$, as none of these formulas are removed by the construction of Γ^*.

The tableaux procedure described in this section gives a decision procedure for $\mathcal{CSL}$. To estimate its complexity, let the length of A, the initial formula, be n. It is not hard to see that any tableau set saturated under blocking may contain at most $O(2^n)$ labels. As matter of fact by the blocking conditions no more than $O(2^n)$ labels can be introduced by dynamic rules $F2 \leftleftarrows$ and $F2\Box_x$. Thus a saturated set under blocking will contain most $O(2^n)$ tableau formulas. We can hence devise a non deterministic procedure that guesses an open tableau set in $O(2^n)$ steps. This shows that our tableau calculus gives a NExpTime decision procedure for $\mathcal{CSL}$. In light of the results contained in [9] our procedure is not optimal, since it is shown that this logic is ExpTime complete. We will study possible optimization (based for instance on caching techniques) in subsequent work.

5 Conclusion

In this paper, we have studied the logic $\mathcal{CSL}$ over minspaces, and we have obtained two main results: first we have provided a direct, sound and complete axiomatisation of this logic. Furthermore, we have defined a tableau calculus, which gives a decision procedure for this logic.

In [4], a tableau algorithm is proposed to handle logics for metric spaces comprising distance quantifiers of the form $\exists^{<a}A$ and alike, where a is positive integer (together with an interior and a closure operator). As observed in [11], the operator $\leftleftarrows$ can be defined in a related logic that allows quantification on

the parameters in distance quantifiers. The methods proposed in [4] make use of an elegant relational translation to handle distance quantifiers with fixed parameters. However, it is not clear if they can be adapted to handle also the concept similarity operator.

There are a number of issues to explore in future research. The decision procedure outlined in the previous section is not guaranteed to have an optimal complexity, so that we can consider how to improve our calculus in order to match this upper bound. Another issue is the extension of our results to symmetric minspaces, and possibly to other classes of models. Finally, since one original motivation of $\mathcal{CSL}$ is to reason about concept similarity in ontologies, and particularly in description logics, we plan to study further its integration with significant languages of this family.

References

1. Alenda, R., Olivetti, N., Schwind, C.: Comparative concept similarity over minspaces: Axiomatisation and tableaux calculus. Technical report (2009), http://arxiv.org/abs/0902.0899
2. Boolos, G.: The Logic of Provability. Cambridge University Press, Cambridge (1993)
3. Giordano, L., Gliozzi, V., Olivetti, N., Schwind, C.: Tableau calculus for preference-based conditional logics: Pcl and its extensions. ACM Trans. Comput. Log. 10(3) (2009)
4. Hustadt, U., Tishkovsky, D., Wolter, F., Zakharyaschev, M.: Automated reasoning about metric and topology. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 490–493. Springer, Heidelberg (2006)
5. Kurucz, A., Wolter, F., Zakharyaschev, M.: Modal logics for metric spaces: Open problems. In: Artëmov, S.N., Barringer, H., d'Avila Garcez, A.S., Lamb, L.C., Woods, J. (eds.) We Will Show Them (2), pp. 193–208. College Publications (2005)
6. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail. Artificial Intelligence 55, 1–60 (1992)
7. Lewis, D.: Counterfactuals. Basil Blackwell Ltd., Malden (1973)
8. Nute, D.: Topics in Conditional Logic. Reidel Publishing Company, Dordrecht (1980)
9. Sheremet, M., Tishkovsky, D., Wolter, F., Zakharyaschev, M.: Comparative similarity, tree automata, and diophantine equations. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS, vol. 3835, pp. 651–665. Springer, Heidelberg (2005)
10. Sheremet, M., Tishkovsky, D., Wolter, F., Zakharyaschev, M.: A logic for concepts and similarity. J. Log. Comput. 17(3), 415–452 (2007)
11. Sheremet, M., Wolter, F., Zakharyaschev, M.: A modal logic framework for reasoning about comparative distances and topology (submitted, 2008)
12. Stalnaker, R.: A theory of conditionals. In: Rescher, N. (ed.) Studies in Logical Theory, American Philosophical Quarterly. Monograph Series, vol. 2, pp. 98–112. Blackwell, Oxford (1968)

A Schemata Calculus for Propositional Logic

Vincent Aravantinos, Ricardo Caferra, and Nicolas Peltier

LIG, CNRS/Grenoble INP,
Bâtiment IMAG C - 220, rue de la Chimie,
38400 Saint Martin d'Hères, France
Vincent.Aravantinos@imag.fr, Ricardo.Caferra@imag.fr,
Nicolas.Peltier@imag.fr

Abstract. We define a notion of *formula schema* handling arithmetic parameters, indexed propositional variables (e.g. P_i) and iterated conjunctions/disjunctions (e.g. $\bigwedge_{i=1}^{n} P_i$, where n is a *parameter*). Iterated conjunctions or disjunctions are part of their syntax. We define a sound and complete (w.r.t. satisfiability) tableaux-based proof procedure for this language. This schemata calculus (called STAB) allows one to capture proof patterns corresponding to a large class of problems specified in propositional logic. Although the satisfiability problem is undecidable for unrestricted schemata, we identify a class of them for which STAB always terminates. An example shows evidence that the approach is applicable to non-trivial practical problems. We give some precise technical hints to pursue the present work.

1 Introduction

The importance of schemata has been recognized in logic since the very beginning (4^{th} century B.C.) and they play a central role in modern mathematical logic: axiom schemata, inference rules schemata, mathematical induction schema,... (see an overview in [1]). The stoics already used modus ponens in its present form, Aristotle and the stoics set out *skeleton arguments* (see e.g. [2]); all this shows how early the notion of 'schema' came into logic.

From a methodological point of view, in order to use schemata in practice, the first step is to try to characterize, as generally as possible, the features defining a schema. In [1] a *schema* (or *scheme*) is a system with 2 components (or 3, or 4 depending whether the underlying language and the set of instances are explicitly mentioned or not) having as a first component a schema-template, i.e. a syntactical construct containing "blanks", "place holders", "dummies" (or other conventional terms) allowing to express several (possibly infinitely many) expressions (including e.g. proof texts). The *instances* are the intended denoted expressions, statements of another language. The second component of a schema specifies constraints on how the blanks are to be filled.

Different forms of schemata have been used by authors, either in propositional logic (see e.g. [3]) or in first order logic to obtain results in proof theory, in particular related to the number of proof lines (see e.g. [4,5,6,7]). Pragmatically, schemata have been successfully used e.g. in solving open questions in

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 32–46, 2009.

equivalential calculus (i.e. the field of formal logic concerned with the notion of equivalence) with OTTER (see [8]).

Coming to automated deduction, though the notion of schema is recognized as an important one, it deserves more applied works in our opinion. Sometimes schemata are not sufficiently emphasized, e.g. in [9] a nice and deep analysis about the challenge of computer mathematics is given. The authors overview the state of the art (by describing and comparing most powerful existing systems in use) but structuring proofs is not explicitly mentioned (maybe this feature can be included in what they call "mathematical style" or "support reasoning with gaps"). In our approach to schemata (see Section 2) it is clear that they can also help to overcome one of the obstacles to the automation of reasoning pointed out in [10], i.e. *the size of deduction steps*.

Problems (theorems) on finite domains can be specified in propositional logic. Most of those (e.g. colouring graphs, Ramsey theory, digital circuits,...) are stated in a "parameterized way", the parameter being the size of the domain[1]. A typical example is the pigeonhole problem which is parameterized by the number of pigeons ($P_{i,j}$ means that the pigeon i is in the hole j):

$$\left(\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n-1} P_{i,j} \right) \wedge \left(\bigwedge_{k=1}^{n-1} \bigwedge_{i \neq j} (\neg P_{i,k} \vee \neg P_{j,k}) \right)$$

It contains iterations ranging on intervals depending on n. Such iterations are ubiquitous in mathematical reasoning. They also frequently occur in constraint programming specifications. If n is instantiated by a natural number then the expression reduces to a propositional formula. Therefore each *instance* of this schema can be (at least theoretically) solved in propositional logic. However, proving that the schema is unsatisfiable (or satisfiable) *for every instance of* n is much harder. This is even out of the scope of first-order logic (see Theorem 1). These problems can be expressed in higher order logics but it is well-known that such languages are less suitable for automation.

Therefore we investigate a particular form of schemata called *iterations* or *iterated schemata* intended to partially capture the activity of specifying in finite domains. Iterations are formulae's conjunctions and disjunctions whose length depends on an arithmetic parameter (e.g. $\bigvee_{i=1}^{n} P_i$). Such objects are part of the syntax (instead of being part of the meta-language). We provide a tableaux-based procedure for reasoning about such schemata. Obviously the proofs containing iterated formulae usually rely on mathematical induction: the idea is to reduce (by applying transformation rules) the problem to the *same problem* but *in a domain of smaller size.*

We chose to concentrate on *propositional* formulae because propositional logic is decidable, thus easier to handle than more expressive languages such as first-order logic. Furthermore, formula schemata commonly appear in various applications of propositional logic, in particular when modeling a circuit (which often

[1] To stay inside propositional logic's expressive limits, *bounded domains* are assumed.

depends on an integer parameter, e.g. the number of bits) or specifying graph properties (e.g. Ramsey's theorem). A concrete example is given in Section 6.

To the best of our knowledge, there are no other works in this direction. Of course, there exist term languages expressive enough to denote iteration schemata as the one mentioned before: for instance the primal grammar [11] $\hat{f}(n) \rightarrow (P(n) \vee \hat{f}(n-1)), \hat{f}(0) \rightarrow \bot$ denotes the iteration[2] $\bigvee_{i=1}^{n} P_i$. However, term schematisation languages do not allow to *reason* on such iterations.

Several procedures have been designed for proving inductive theorems (see e.g. [14,15,16,17,18,19]). But most systems concentrate on universal quantification, where we have to handle both iterated conjunctions (which can be interpreted as universal quantification on a bounded domain) and iterated disjunctions (i.e. existential quantifications). Adding existential quantification in inductive theorem proving is known to be a difficult problem.

Our work shares some similarities with the ones on Satisfiability Modulo Theory (see e.g. [20,21]). However, instead of extending propositional logic with some (decidable) theories, we consider arithmetic indexes and bounded quantification over these indexes. The obtained formulae are non ground since they contain index variables. Clearly, the combination of these two lines of research deserves to be investigated (to handle schemata of SMT problems).

The rest of the paper is structured as follows. In Section 2 we introduce a formal language for reasoning on schemata (syntax and semantics). This extends propositional logic by allowing indexed variables and disjunctions/conjunctions ranging on intervals (with symbolic bounds). We show that the satisfiability problem is only semi-decidable. In Section 3 we design a tableau procedure, called STAB (for ***s**chemata **tab**leaux*), for this language: we extend the usual tableaux for propositional logic [22,23] to handle schemata symbolically (avoiding systematic instantiation of the parameters). We prove its soundness and completeness w.r.t. satisfiability (no refutationally complete procedure exists, as shown in Section 2). In Section 4 we provide some useful extensions to STAB. In Section 5 we introduce a class of schemata for which STAB always terminates, yielding a decision procedure for this class. In Section 6 we show a simple example of application. Section 7 contains a short conclusion and some lines of future work.

Due to space restriction, detailed proofs are omitted.

2 Schemata of Propositional Formulae

2.1 Syntax

In all the following indexed propositions are written $P_i, P_{i,j}, P_{i,j,k}, \ldots$ and integer variables are written $i, j, k, \ldots$ or $n, p, q, \ldots$ Schemata are denoted by $S, S_1, \ldots,$ sets of schemata by $\mathcal{S}$, interpretations by $\mathcal{I}, \mathcal{J}$. Tableaux are written $\mathcal{T}, \mathcal{T}_0, \ldots$ and nodes in a tableau $\alpha, \beta, \ldots$

[2] It is worth mentioning that this iteration cannot be denoted by other term schematisation languages [12,13] because the inductive context is not constant.

Definition 1 (Integer terms). *Let $\mathcal{V}$ be an infinite set of* integer variables. *The set of* integer terms*, written $\mathcal{T}_{\mathcal{N}}$ is the smallest set containing $\mathcal{V}$, $\mathbb{Z}$ and s.t. for all $k \in \mathbb{Z}$, $t_1, t_2 \in \mathcal{T}_{\mathcal{N}}$: $t_1 + t_2 \in \mathcal{T}_{\mathcal{N}}$ and $k \times t_1 \in \mathcal{T}_{\mathcal{N}}$. For all $t \in \mathcal{T}_{\mathcal{N}}$, $\mathcal{V}\mathrm{ar}(t)$ is the set of variables that occur in t. A* ground term *is a term t s.t. $\mathcal{V}\mathrm{ar}(t) = \emptyset$.*

We consider linear integer terms for the sake of simplicity, more expressive arithmetic expressions can be considered provided they belong to decidable classes.

Definition 2 (Indexed propositions). *Let $(\mathcal{P}_k)_{k \in \mathbb{N}}$ be a family of symbols. For all $k \in \mathbb{N}$, $P \in \mathcal{P}_k$, and $t_1, \ldots, t_k \in \mathcal{T}_{\mathcal{N}}$, $P_{t_1,\ldots,t_k}$ is an* indexed proposition. *$t_1, \ldots, t_k$ are the* indices *of $P_{t_1,\ldots,t_k}$. An indexed proposition $P_{t_1,\ldots,t_k}$ s.t. $t_1, \ldots, t_k \in \mathbb{Z}$ is called a* propositional variable*; a propositional variable or its negation is a* literal.

Definition 3 (Schemata). *The set of* formula schemata *is the smallest set s.t.*

- *$\top$, $\bot$ are formula schemata.*
- *Each indexed proposition is a formula schema.*
- *If S_1, S_2 are schemata then $S_1 \vee S_2$, $S_1 \wedge S_2$ and $\neg S_1$ are formula schemata.*
- *If S is a formula schema, $t_1, t_2 \in \mathcal{T}_{\mathcal{N}}$, and i is a variable, then $\bigwedge_{i=t_1}^{t_2} S$ and $\bigvee_{i=t_1}^{t_2} S$ are formula schemata (such schemata are called* iterations*).*

Example 1.

$$S = Q_1 \wedge \bigwedge_{i=1}^{n} \left(P_i \wedge \bigvee_{j=1}^{n+1} \neg Q_j \vee Q_{j+1} \right) \quad \text{is a formula schema.}$$

Q_1, P_i, Q_j and Q_{j+1} are indexed propositions. $\bigwedge_{i=1}^{n} \left(P_i \wedge \bigvee_{j=1}^{n+1} \neg Q_j \vee Q_{j+1} \right)$ and $\bigvee_{j=1}^{n+1} \neg Q_j \vee Q_{j+1}$ are the only iterations occurring in S.

A variable i is *bound* in S if S contains an iteration of the form $\mathit{\Pi}_{i=a}^{b} S'$ ($\mathit{\Pi} \in \{\bigvee, \bigwedge\}$), it is *free* (or is a *parameter* of S) if it occurs in S but not in the scope of an iteration $\mathit{\Pi}_{i=a}^{b} S'$. *Substitutions* on integer variables are defined as usual. We write $[t_1/i_1, \ldots, t_k/i_k]$ for the substitution mapping resp. $i_1, \ldots, i_k$ to $t_1, \ldots, t_k$. We now assume that no variable of any schema S can be simultaneously free and bound in S, and if $\mathit{\Pi}_{i=a}^{b} S_i$ and $\mathit{\Sigma}_{j=c}^{d} S'_j$ (where $\mathit{\Pi}, \mathit{\Sigma} \in \{\bigvee, \bigwedge\}$) are two distinct iterations occurring in S then i and j are distinct. Such a schema is said *rectified.*

2.2 Semantics

Definition 4 (Semantics). *An* interpretation of the schemata language *is a function mapping every propositional variable to a truth value* **T** *or* **F** *and every integer variable to an integer. Then the* semantic $[\![S]\!]_{\mathcal{I}}$ *of a propositional schema in an interpretation $\mathcal{I}$ is inductively defined as:*

- $[\![\top]\!]_{\mathcal{I}} = \mathbf{T}$, $[\![\bot]\!]_{\mathcal{I}} = \mathbf{F}$
- $[\![P_{t_1,\ldots,t_k}]\!]_{\mathcal{I}} = \mathcal{I}(P_{\mathcal{I}(t_1),\ldots,\mathcal{I}(t_k)})$ *where the interpretation of arithmetic expressions is defined as usual.*
- $[\![\neg\Phi]\!]_{\mathcal{I}} = \mathbf{T}$ *iff* $[\![\Phi]\!]_{\mathcal{I}} = \mathbf{F}$.
- $[\![\Phi \vee \Phi']\!]_{\mathcal{I}} = \mathbf{T}$ *iff* $[\![\Phi]\!]_{\mathcal{I}} = \mathbf{T}$ *or* $[\![\Phi']\!]_{\mathcal{I}} = \mathbf{T}$.
- $[\![\Phi \wedge \Phi']\!]_{\mathcal{I}} = \mathbf{T}$ *iff* $[\![\Phi]\!]_{\mathcal{I}} = \mathbf{T}$ *and* $[\![\Phi']\!]_{\mathcal{I}} = \mathbf{T}$.
- $[\![\bigvee_{i=t_1}^{t_2} S]\!]_{\mathcal{I}} = \mathbf{T}$ *iff there is an integer* k *s.t.* $\mathcal{I}(t_1) \leq k \leq \mathcal{I}(t_2)$ *holds and* $[\![S]\!]_{\mathcal{J}} = \mathbf{T}$ *where* $\mathcal{J}$ *is s.t.* $\mathcal{J}(i) = k$ *and* $\mathcal{J}(j) = \mathcal{I}(j)$ *for* $j \neq i$.
- $[\![\bigwedge_{i=t_1}^{t_2} S]\!]_{\mathcal{I}} = \mathbf{T}$ *iff for every integer* k *s.t.* $\mathcal{I}(t_1) \leq k \leq \mathcal{I}(t_2)$ *holds:* $[\![S]\!]_{\mathcal{J}} = \mathbf{T}$ *where* $\mathcal{J}$ *is defined the same way as for* $\bigvee$.

A schema S *is* satisfiable *iff there is an interpretation* $\mathcal{I}$ *s.t.* $[\![S]\!]_{\mathcal{I}} = \mathbf{T}$. *Then* $\mathcal{I}$ *is called a* model *of* S.

It is trivially semi-decidable to know if a schema is satisfiable:

Proposition 1. *The set of satisfiable schemata is recursively enumerable.*

Proof. Let S be a schema. S is satisfiable iff there is a ground substitution σ of the parameters of S s.t. $S\sigma$ is satisfiable. The set of ground substitutions is enumerable. Moreover, $S\sigma$ can be easily turned into an equivalent propositional formula (in the usual sense): as σ is ground, the bounds of the iterations in S are always finite, thus they can be unfolded and replaced by standard conjunctions or disjunctions. Thus it is decidable whether $S\sigma$ is satisfiable or not. □

However this is not more than semi-decidable as shows the following result:

Theorem 1. *The set of satisfiable schemata is not recursive.*

Proof. (Sketch) The proof is by reduction to Post's correspondence problem. Let $\mathcal{A} = \{c_1, \ldots, c_K\}$ be a finite alphabet and $u_1, \ldots, u_P$ and $v_1, \ldots, v_P$ two finite lists of non-empty words over $\mathcal{A}$. A *solution* is a sequence of indices $(sol_k)_{k=1..N}$ s.t. $N \geq 1$ and $u_{sol_1} \ldots u_{sol_N} = v_{sol_1} \ldots v_{sol_N}$. $u_{sol_1} \ldots u_{sol_N}$ is called the *solution witness*. We show how to encode any instance of the problem into a schema S so that S is satisfiable iff this instance has a solution. More precisely we construct S of parameter n s.t. for all $N \in \mathbb{N}$, $S[N/n]$ is satisfiable iff there is a solution of length N.

The lists u and v are easily encoded through indexed propositions as well as the solution sequence *sol*. The main work is to schematise the fact that the solution sequence is indeed a solution. For this, we introduce an indexed proposition that checks if this is the case until the i^{th} character of the solution witness. $Check_{w_1,p_1,w_2,p_2,i}$ is this proposition, specified by induction on i: if $Check_{w_1,p_1,w_2,p_2,i}$ holds and if the character at position p'_1 of the word $u_{sol_{w'_1}}$ is equal to the character at position p'_2 of the word $v_{sol_{w'_2}}$ and to the $i+1^{th}$ character of the solution witness then $Check_{w'_1,p'_1,w'_2,p'_2,i+1}$ holds. w'_1, w'_2, p'_1, p'_2 depend on both current positions in words from u and v. There are four possible combinations. We just give one: if p_1 is the last position of the word $v_{sol_{w_1}}$ and p_2 is *not* the last position of $v_{sol_{w_2}}$ then $w'_1 = w_1 + 1$, $p'_1 = 1$, $w'_2 = w_2$ and

$p'_2 = p_2 + 1$ i.e. we go on to the first character of the next word in u and to the next character of the current word in v. This is formally expressed by:

$$\bigwedge_{w_1=1}^{n} \bigwedge_{w_2=1}^{n} \bigwedge_{p_1=1}^{M} \bigwedge_{p_2=1}^{M} \bigwedge_{i=1}^{M\times n} (Check_{w_1,p_1,w_2,p_2,i} \wedge LastCharU_{w_1,p_1}$$
$$\wedge \neg LastCharV_{w_2,p_2} \wedge Eq_{w_1+1,1,w_2,p_1+1}) \Rightarrow Check_{w_1+1,1,w_2,p_2+1,i+1}$$

where M is the maximum of the lengths of all words in both lists (M is a constant for a given instance of Post's problem), $LastCharU_{w,p}$ is true iff p is the position of the last character of u_{sol_w} and Eq_{w_1,p_1,w_2,p_2} is true iff the characters at respective positions p_1, p_2 in $u_{sol_{w_1}}, v_{sol_{w_2}}$ are equal.

All those schemata and the ones that remain to express the whole induction are defined easily. We finally express that there is a rank where both sides are equal and where both positions are the last of their respective words. □

3 A Proof Procedure: STAB

We provide now a set of block tableaux rules [22] that ensure completeness w.r.t. satisfiability (we know from Theorem 1 that we cannot ensure refutational completeness). Those rules are concise and natural, and, compared to the naive procedure described in the proof of Proposition 1, STAB is much more efficient and terminates more often (see the end of Section 3.1). However, its main interest is that it is much better suited for termination when dealing with unsatisfiable schemata, as will be clear from the extensions defined in Section 4.

3.1 Inference Rules

STAB works with both schemata *and* constraints on the parameters:

Definition 5 (Parameter constraints). *The set of* parameter constraints *on a schema S is the set of first-order formulae of atoms $t_1 \bullet t_2$ where $\bullet \in \{=, <, >, \leq, \geq\}$ and $t_1, t_2 \in \mathcal{T}_{\mathcal{N}}$ s.t. $\mathcal{V}\mathrm{ar}(t_1)$ and $\mathcal{V}\mathrm{ar}(t_2)$ contain only parameters of S.*

Definition 6 (Tableau). *A* tableau *is a tree $\mathcal{T}$ s.t. each node α occurring in $\mathcal{T}$ is labeled by a set $S_{\mathcal{T}}(\alpha)$ containing schemata and parameter constraints.*

As usual a tableau is generated from another tableau by applying some extension rules. Let $r = \dfrac{\mathcal{P}}{C_1|\ldots|C_k}$ be a rule where $\mathcal{P}$ denotes a set of schemata and constraints, and $C_1, \ldots, C_k$ denote the conclusions of the rule. Let α be a leaf of a tree $\mathcal{T}$. If a subset $\mathcal{S}$ of $S_{\mathcal{T}}(\alpha)$ matches $\mathcal{P}$ then we can *extend* the tableau by adding k children to α, each of them labeled with $C_i\sigma \cup S_{\mathcal{T}}(\alpha) \setminus \mathcal{S}$ where σ is the matching substitution. Notice that, with this definition, *each leaf contains all schemata and constraints in the branch.* A leaf is *closed* iff its parameter constraints are unsatisfiable (this can be detected using decision procedures for arithmetic without multiplication see e.g. [24]). Rules of STAB are defined as follows:

Definition 7 (Extension rules). *We assume (w.l.o.g) that schemata are in negative normal form. The* extension rules *are:*

- *The usual rules of propositional tableaux:*

$$(\wedge):\ \frac{A \wedge B}{A \quad B} \qquad (\vee):\ \frac{A \vee B}{A \mid B}$$

- *Rules proper to schemata ("iteration rules")*[3]*:*

$$(\text{Iterated } \wedge):\ \frac{\bigwedge_{i=a}^{b} S}{\begin{array}{c|c} b \geq a & b < a \\ \bigwedge_{i=a}^{b-1} S \wedge S[b/i] & \top \end{array}} \qquad (\text{Iterated } \vee):\ \frac{\bigvee_{i=a}^{b} S}{\begin{array}{c} b \geq a \\ \bigvee_{i=a}^{b-1} S \vee S[b/i] \end{array}}$$

- *The closure rule adds the constraints needed for the branch* not to be closed*:*

$$(\text{Closure}):\ \frac{P_{t_1,\ldots,t_n} \quad \neg P_{s_1,\ldots,s_n}}{t_1 \neq s_1 \vee \ldots \vee t_n \neq s_n}$$

Generalising the iteration rules. The two rules on iterations could be more general. Indeed we could define the following rules schema:

$$\frac{\bigwedge_{i=a}^{b} S}{\begin{array}{c|c} b \geq a & b < a \\ \bigwedge_{i=a}^{j-1} S \wedge S[j/i] \wedge \bigwedge_{i=j+1}^{b} S & \top \end{array}} \qquad \frac{\bigvee_{i=a}^{b} S}{\begin{array}{c} b \geq a \\ \bigvee_{i=a}^{j-1} S \vee S[j/i] \vee \bigvee_{i=j+1}^{b} S \end{array}}$$

where one has to specify how j is chosen. In the present paper $j = b$ (then the part $\mathit{\Pi}_{i=j+1}^{b} S$ of the iteration is empty, and thus removed). Other strategies are possible (but in order to ensure soundness we must have $a \leq j \leq b$): one can choose a term j allowing further applications of the closure rule e.g. if the branch contains $\bigwedge_{i=1}^{2n} P_i$ and $\neg P_n$ then the extension rule would apply with $j = n$ (easily automated). A third possibility is simply to add a new parameter j in which case the constraint $a \leq j \leq b$ must be added. Such a strategy even allows further simplifications e.g. in the Iterated $\vee$ rule, if we know that $S[j/i]$ holds we can "cut" immediately the two (future) branches dealing with $\bigvee_{i=a}^{j-1} S$ and $\bigvee_{i=j+1}^{b} S$.

STAB without the upcoming extensions is already better than the naive procedure. First it terminates in some cases where the schema is unsatisfiable (whereas the naive procedure never terminates in such a case, unless the schema is just an unsatisfiable propositional formula). This is trivially the case for any schema $\bigwedge_{i=1}^{n} F$ with $n \geq 1$, where F is propositionally unsatisfiable. Second, it can find a model much faster than the naive procedure. Consider e.g. $(\bigwedge_{i=n}^{10000} P) \wedge (\neg P \vee F)$ where F is an unsatisfiable formula. In this case STAB immediately finds the model $n > 10000$ and $P = \mathbf{F}$.

[3] The right branch in the conclusion of the Iterated $\wedge$ rule is required, e.g. to detect that $\bigwedge_{i=1}^{n} \bot$ is satisfiable with $n = 0$, of course, the formula $\top$ could be removed.

3.2 Soundness and Completeness

Definition 8 (Tableau semantics). *For every node α in a tableau $\mathcal{T}$, $S_{\mathcal{T}}(\alpha)$ is interpreted as the conjunction of its elements. $\mathcal{T}$ is satisfied in an interpretation $\mathcal{I}$ if there exists a leaf α in $\mathcal{T}$ s.t. $\mathcal{I} \models S_{\mathcal{T}}(\alpha)$. In such a case we write $\mathcal{I} \models \mathcal{T}$.*

Lemma 1. *If $\mathcal{T}'$ is a tableau obtained by applying one of the extension rules on a leaf α of a tableau $\mathcal{T}$ then $\mathcal{I} \models S_{\mathcal{T}}(\alpha)$ iff there exists a leaf β of $\mathcal{T}'$ s.t. β is a child of α in $\mathcal{T}'$ and $\mathcal{I} \models S_{\mathcal{T}'}(\beta)$ (i.e. the rules are sound and invertible).*

Proof. (Sketch) By inspection of the extension rules. □

A leaf is *irreducible* if no extension rule applies to it.

Lemma 2. *If a leaf α in $\mathcal{T}$ is irreducible and not closed then $\mathcal{T}$ is satisfiable.*

Proof. (Sketch) Let Φ be the set of arithmetic constraints in $S_{\mathcal{T}}(\alpha)$ and $\mathcal{S} \stackrel{\text{def}}{=} S_{\mathcal{T}}(\alpha) \setminus \Phi$. As α is not closed Φ is satisfiable, let σ be a solution of Φ. All the schemata in $\mathcal{S}$ are literals (otherwise decomposition rules apply) and $c_{\mathcal{T}}(\alpha) = 0$ (otherwise the closure rule applies) where $c_{\mathcal{T}}(\alpha)$ is the number of pairs $P_{t_1,\dots,t_n}$, $\neg P_{s_1,\dots,s_n} \in S_{\mathcal{T}}(\alpha)$ s.t. there is an interpretation $\mathcal{I}$ s.t. for all $i \in [1..n]$, $[\![t_i]\!]_{\mathcal{I}} = [\![s_i]\!]_{\mathcal{I}}$. Hence $\mathcal{S}$ is a set of ground literals not containing two pairs of complementary literals. Thus $S_{\mathcal{T}}(\alpha)\sigma$ is satisfiable and by definition $\mathcal{T}$ is satisfiable. □

Theorem 2 (Soundness). *Let $\mathcal{T}$ be a tableau. If a tableau $\mathcal{T}'$ is obtained from $\mathcal{T}$ by application of the extension rules, and if $\mathcal{T}'$ contains an irreducible and not closed leaf then $\mathcal{T}$ is satisfiable.*

Proof. This follows immediately from Lemmata 1 and 2. □

We now prove that the procedure is complete w.r.t. satisfiability. Let $\mathcal{I}$ be an interpretation and S a formula. We define $m_{\mathcal{I}}(S)$ as follows:

- $m_{\mathcal{I}}(F) \stackrel{\text{def}}{=} 0$ if F is a parameter constraint.
- $m_{\mathcal{I}}(P) \stackrel{\text{def}}{=} 1$ if P is an indexed proposition or its negation, or P is $\top$ or $\bot$.
- $m_{\mathcal{I}}(S_1 \star S_2) \stackrel{\text{def}}{=} m_{\mathcal{I}}(S_1) + m_{\mathcal{I}}(S_2)$ if $\star \in \{\vee, \wedge\}$.
- $m_{\mathcal{I}}(\Pi_{i=a}^{b} S) \stackrel{\text{def}}{=} 2$ if $[\![b]\!]_{\mathcal{I}} < [\![a]\!]_{\mathcal{I}}$
- $m_{\mathcal{I}}(\Pi_{i=a}^{b} S) \stackrel{\text{def}}{=} l - k + 2 + \Sigma_{j=k}^{l} m_{\mathcal{J}_j}(S)$ else, where $\Pi \in \{\bigwedge, \bigvee\}$, $k = [\![a]\!]_{\mathcal{I}}$, $l = [\![b]\!]_{\mathcal{I}}$ and $\mathcal{J}_j$ is an interpretation defined exactly as $\mathcal{I}$, except that $[\![i]\!]_{\mathcal{J}_j} \stackrel{\text{def}}{=} j$.

If $\mathcal{S}$ is a set, then $m_{\mathcal{I}}(\mathcal{S}) \stackrel{\text{def}}{=} \{m_{\mathcal{I}}(S) \mid S \in \mathcal{S}\}$. If $\mathcal{T}$ is a tableau and α is a leaf in $\mathcal{T}$ then $m_{\mathcal{I}}(\alpha, \mathcal{T}) \stackrel{\text{def}}{=} (m_{\mathcal{I}}(S_{\mathcal{T}}(\alpha)), c_{\mathcal{T}}(\alpha))$ where $c_{\mathcal{T}}(\alpha)$ is defined in the proof of Lemma 2. This measure is ordered using the multiset and lexicographic extensions of the usual ordering on natural numbers.

Lemma 3. *Let $\mathcal{I}$ be an interpretation. Let $\mathcal{T}$ be a tableau. If $\mathcal{T}'$ is deduced from $\mathcal{T}$ by applying an extension rule on a leaf α s.t. $\mathcal{I} \models S_{\mathcal{T}}(\alpha)$, then for every child β of α in $\mathcal{T}'$ s.t. $\mathcal{I} \models S_{\mathcal{T}'}(\beta)$, we have $m_{\mathcal{I}}(\beta, \mathcal{T}') < m_{\mathcal{I}}(\alpha, \mathcal{T})$.*

Proof. (Sketch) All the rules except the iteration rule and the closure rule replace a formula by simpler ones, hence it is easy to see that $m_{\mathcal{I}}(S_{\mathcal{T}}(\alpha))$ decreases. The iteration rules replace an iteration of length l either by $\top$ or by a disjunction/conjunction of an iterated disjunction/conjunction of length $l-1$, and a smaller formula. Since $l > l-1$, $m_{\mathcal{I}}(S_{\mathcal{T}}(\alpha))$ decreases. The closure rule does not affect $m_{\mathcal{I}}(S_{\mathcal{T}}(\alpha))$ but obviously decreases $c_{\mathcal{T}}(\alpha)$. □

A *derivation* is a (possibly infinite) sequence of tableaux $(\mathcal{T}_i)_{i \in I}$ s.t. I is either $[0..n]$ for some $n \geq 0$, or $\mathbb{N}$ and s.t. for all $i \in I \setminus \{0\}$, $\mathcal{T}_i$ is obtained from $\mathcal{T}_{i-1}$ by applying one of the rules. A derivation is *fair* if there is $i \in I$ s.t. $\mathcal{T}_i$ contains an irreducible not closed leaf or if for all $i \in I$ and every not closed leaf α in $\mathcal{T}_i$ there is $j \geq i$ s.t. a rule is applied on α in $\mathcal{T}_j$ (i.e. no leaf can be "freezed").

Theorem 3 (Completeness w.r.t. satisfiability). *Let $\mathcal{T}_0$ be a satisfiable tableau and let $\mathcal{I}$ be a model of $\mathcal{T}_0$. If $(\mathcal{T}_n)_{n \in I}$ is a fair derivation then there is $k \in I$ and a leaf α_k in $\mathcal{T}_k$ s.t. α_k is irreducible and not closed.*

Proof. By Lemma 1, for all $i \in I$, $\mathcal{T}_i$ contains a leaf α_i s.t. $\mathcal{I} \models S_{\mathcal{T}_i}(\alpha_i)$. Let $k \in I$ s.t. $m_{\mathcal{I}}(\alpha_k, \mathcal{T}_k)$ is minimal (k exists since $m_{\mathcal{I}}(\alpha_i, \mathcal{T}_i)$ is well-founded). Assume a rule is applied on α_k in the derivation, on some tableau $\mathcal{T}_l$. By Lemma 1 there is a child β of α_k s.t. $\mathcal{I} \models S_{\mathcal{T}_l}(\beta)$. By Lemma 3 we have $m_{\mathcal{I}}(\beta, \mathcal{T}_l) < m_{\mathcal{I}}(\alpha_k, \mathcal{T}_k)$ which is impossible. Thus no rule is applied on α_k. Since the derivation is fair, α_k is irreducible (or there is another leaf that is irreducible). □

4 Extensions

STAB is intuitive and complete for satisfiability but it rarely terminates. The extensions significantly extend the class of formulae that STAB is able to refute.

4.1 Infinite Iterations (Looping)

The reason for STAB not to terminate is that an iteration is infinitely unfolded by the iteration rules. Assume for instance that S is a propositional unsatisfiable formula. Then starting from $\bigvee_{i=1}^{n} S$ one could derive an infinite sequence of formulae of the form $\bigvee_{i=1}^{n-1} S, \ldots, \bigvee_{i=1}^{n-k} S$, for every $k \in \mathbb{N}$. Detecting looping is the most natural way to avoid this divergence: if, while extending the tableau, we find a schema that has already been seen, e.g. up to a shift of arithmetic variables, then there is no need to consider it a second time and we can stop the procedure. This is a procedural view of an induction proof.

We give a first definition of looping, powerful enough to ensure termination for the class of schemata considered in Section 5 (it is useless to look for the most general definition which theoretically does not exist). *We assume all parameters are interpreted as positive integers.* This can be specified by adding at the root of the tableau the constraint $n \geq 0$ for every parameter n.

Definition 9 (Looping). *Let α, β be two nodes of a tableau $\mathcal{T}$. Let $n_1, \ldots, n_k$ be the free variables of $S_{\mathcal{T}}(\alpha)$. Then β* loops *on α if there are $p_1, \ldots, p_k \in \mathbb{N}$ s.t. one at least is positive and every model of $S_{\mathcal{T}}(\beta)$ is a model of $S_{\mathcal{T}}(\alpha)[n_1 - p_1/n_1, \ldots, n_k - p_k/n_k]$.*

When a leaf loops, it is treated as a closed branch (though it is not necessarily unsatisfiable), we say that it is *blocked*. Notice that α and β may be on different branches, thus looping may occur more often, allowing more simplifications.

Theorem 2 trivially remains true but the proof of Theorem 3 must be adapted:

Theorem 4 (Completeness w.r.t. satisfiability). *Let $\mathcal{T}_0$ be a satisfiable tableau and $\mathcal{I}$ be a model of $\mathcal{T}_0$. If $(\mathcal{T}_n)_{n\in I}$ is a fair derivation then there exist $k \in I$ and a leaf α_k in $\mathcal{T}_k$ s.t. α_k is irreducible and neither closed nor blocked.*

Proof. (Sketch) We suppose there is only one parameter n for the sake of simplicity and leave the general case to the reader. Let S' be a leaf looping on a node S. Assume w.l.o.g. that $\mathcal{I}$ be s.t. $\mathcal{I}(n)$ is minimal. Rather than repeating the whole proof of Theorem 3, we only check that the addition of the new looping rule does not destroy completeness (i.e. no model is missed). The only possibility of missing a model would be to block a looping branch s.t. this makes us miss a (lower) possibly irreducible and not closed branch. If $\mathcal{I} \models S'$ then $\mathcal{I} \models S[n-k/n]$ and hence $\mathcal{J} \models S$ where $\mathcal{J}$ is identical to $\mathcal{I}$ except that $\mathcal{J}(n) = \mathcal{I}(n) - k$. Thus by Lemma 1, $\mathcal{J}$ is a model of the root i.e. for $\mathcal{T}_0$; as $k > 0$, this is a contradiction with the fact that $\mathcal{I}$ is minimal. Thus $\mathcal{I} \not\models S'$. Hence $\mathcal{I} \not\models S''$ for all children S'' of S' (by Lemma 1 converse), so indeed *no model is missed.* □

To apply the looping rule in practice one has to compute the natural numbers $p_1, \ldots, p_k$ and check that the implication holds. This problem is obviously undecidable, but it is possible to define a stronger (decidable) relationship between S and S' ensuring that $p_1, \ldots, p_k$ exist. The underlying idea is the following: let be $S = \bigvee_{i=a}^{b} S_i$ and $S' = \bigvee_{j=c}^{d} S_j$. To check that $S \Rightarrow S'$, it is sufficient that for all $i \in [a..b]$ there is $j \in [c..d]$ s.t. $S_i \Rightarrow S_j$ is verified. If S_i, S_j are indexed propositions then $S_i \Rightarrow S_j$ holds if $i = j$ (so the above condition is equivalent to $[a..b] \subseteq [c..d]$). Formally we inductively construct an arithmetic formula (without multiplication) from the structure of both schemata for which we want to check the looping. This can be seen as a form of subsumption between schemata.

Definition 10. *Let S, S' be two schemata or constraints. We inductively define the arithmetic formula $F_{S\Rightarrow S'}$ as follows:*

- $F_{S\Rightarrow S'} \stackrel{def}{=} S \Rightarrow S'$ *if S, S' are constraints.*
- $F_{S_1\vee S_2\Rightarrow S'} \stackrel{def}{=} F_{S\Rightarrow S'_1\wedge S'_2} \stackrel{def}{=} F_{S_1\Rightarrow S'} \wedge F_{S_2\Rightarrow S'}$
- $F_{S_1\wedge S_2\Rightarrow S'_1} \stackrel{def}{=} F_{S\Rightarrow S'_1\vee S'_2} \stackrel{def}{=} F_{S_1\Rightarrow S'} \vee F_{S_2\Rightarrow S'}$
- $F_{\bigvee_{i=a}^{b} S\Rightarrow S'} \stackrel{def}{=} F_{S\Rightarrow \bigwedge_{i=a}^{b} S'} \stackrel{def}{=} \forall i \in [a..b] \cdot F_{S\Rightarrow S'}$
- $F_{\bigwedge_{i=a}^{b} S\Rightarrow S'} \stackrel{def}{=} F_{S\Rightarrow \bigvee_{i=a}^{b} S'} \stackrel{def}{=} \exists i \in [a..b] \cdot F_{S\Rightarrow S'}$
- $F_{\neg S\Rightarrow \neg S'} \stackrel{def}{=} F_{S'\Rightarrow S}$ *if S, S' are non arithmetic atoms.*
- $F_{p_{t_1,\ldots,t_n}\Rightarrow p_{s_1,\ldots,s_n}} \stackrel{def}{=} (t_1 = s_1 \wedge \ldots \wedge t_n = s_n)$
- $F_{S\Rightarrow S'} \stackrel{def}{=} \bot$ *otherwise.*

Then checking that the looping rule is applicable between a leaf α and a node α' in a tableau $\mathcal{T}$, amounts to checking that the arithmetic formula $\exists p_1, \ldots, p_k \cdot$

$p_1,\ldots,p_k > 0 \wedge \forall n_1,\ldots,n_k \cdot F_{S_\mathcal{T}(\alpha)\Rightarrow S_\mathcal{T}(\alpha')[n_1-p_1,\ldots,n_k-p_k]}$ is valid (sets of schemata are interpreted as conjunctions). This follows from Proposition 2 which is proved by an easy induction on S, S':

Proposition 2. *Every model of both $F_{S\Rightarrow S'}$ and S is a model of S'.*

4.2 Purity Principle

The pure literal rule is standard in propositional theorem proving. It consists in evaluating a literal L to $\top$ in a formula S (in nnf) if the complement of L does not occur in S. Such a literal is called *pure*. It is well-known that this operation preserves satisfiability and may allow many simplifications.

We show how to extend the pure literal rule to schemata. The conditions on L have to be strengthened in order to take iterations into account. For instance, if $L = P_n$ and S contains $\bigvee_{i=1}^{2n} \neg P_i$ then L is not pure in S, since $\neg P_i$ is the complement of L for $i = n$ (and $1 \leq n \leq 2n$). On the other hand P_{2n+1} may be pure in S (since $2n + 1 \not\in [1..2n]$).

Let S be a rectified schema. We write $IC(S)$ for the conjunction of arithmetic constraints of the form $a \leq i \wedge i \leq b$, s.t. S contains an iteration $\mathit{\Pi}_{i=a}^{b} S_i$.

Definition 11 (Pure literal). *A literal $P_{t_1,\ldots,t_n}$ (resp. $\neg P_{t_1,\ldots,t_n}$) is* pure *in S if for every occurrence of a literal $\neg P_{s_1,\ldots,s_n}$ (resp. $P_{s_1,\ldots,s_n}$) in S, the arithmetic formula $IC(S) \wedge t_1 = s_1 \wedge \ldots \wedge t_n = s_n$ is unsatisfiable.*

Proposition 3. *Let L be a pure literal in a rectified schema S then S is satisfiable iff the schema obtained by substituting $\top$ to L in S is satisfiable.*

The pure literal rule applies this substitution, it may ease the application of the looping rule by removing redundant literals (see the proof of Theorem 5).

5 A Terminating Class

Termination cannot be ensured for schemata in general (Theorem 1). However it can be guaranteed for some useful syntactic classes of schemata.

A schema S is *flat* if for every iteration $\mathit{\Pi}_{i=a}^{b} S_i$ occurring in S, S_i does not contain any iteration (i.e. iterations cannot be nested in S). S is *aligned* on $[a..b]$ if all iterations occurring in S are of the form $\mathit{\Pi}_{i=a}^{b} S_i$. S is *of limited propagation* if there are $l_1, l_2 \in \mathbb{Z}$ s.t. for every indexed proposition that occurs in an iteration $\mathit{\Pi}_{i=a}^{b} S_i$, each of its indices is of the form $i + c$ for some $c \in [l_1..l_2]$; l_1, l_2 are called the *propagation limits.*

Definition 12. *A schema is* regular *if it has a unique parameter n and if it is flat, of limited propagation and aligned on $[k..n-l]$ for some $k, l \in \mathbb{N}$.*

Though being a simple class, regular schemata allow to specify, e.g. many parameterized circuits. The main limitation is that nesting of iterations is disallowed. We consider the following strategy τ for applying the extension rules:

- The propositional extension rules, the looping, closure and pure literal rules are applied as soon as possible on all leaves, with the highest priority.
- The iteration rules are applied only on iterations of maximal length (w.r.t. the natural partial ordering on arithmetic expressions).

Theorem 5. *τ terminates on every regular schema.*

Proof. (Sketch) Let be $l_1, l_2 \in \mathbb{Z}$, $k, l \in \mathbb{N}$ and S a regular schema aligned on $[k..n-l]$, of propagation limits l_1, l_2. Assume that an infinite branch is constructed. By definition of the strategy, after some time, the m last ranks of every iteration have been removed. Thus all the remaining iterations are of the form $\Pi_{i=k}^{n-l-m} S_i$ and we have the arithmetic constraint $n-l-m-k+1 \geq 0$, i.e. $n \geq l+m+k-1$. Moreover, we assume that all the formulae occurring in the branch at this step are irreducible by the first set of rules, hence they are either literals or iterations.

Literals occurring in the branch, but not in the scope of an iteration, are either literals of S or literals introduced by previous applications of the iteration rules. The former are indexed by expressions of the form $u \times n + v$ for some $u, v \in \mathbb{Z}$ and the latter by $n-l-m+c$, where $i < m$ and $c \in [l_1+1..l_2+m]$.

If a literal is indexed by an expression $u \times n + v$ that is outside $[k..n-l-m]$, then it must be pure in every iteration, hence (by irreducibility w.r.t. the closure rule) must be pure in the branch. However, if m is big enough then, by the above arithmetic constraints, $u \times n + v$ cannot be in $[k..n-l-m+l_2]$ if $u \neq 0$: if u is negative, then it suffices to take $m > (k-v)/u - l - k + 1$ to ensure $u \times n + v < k$, otherwise $m \geq l_2 - l - v \Rightarrow u \times n + v > n - l - m + l_2$. Thus every literal indexed by integer terms of this form may be removed by the pure literal rule.

Similarly literals indexed by expressions of the form $n-l-m+c$ where $c > l_2$ are deleted by the extension rules, thus we may assume that $c \in [l_1+1..l_2]$. Clearly, there is a *finite* number of such schemata up to a translation on n. Hence the looping rule applies at some point in every branch, and τ terminates. □

6 Example: The N-Bit Adder

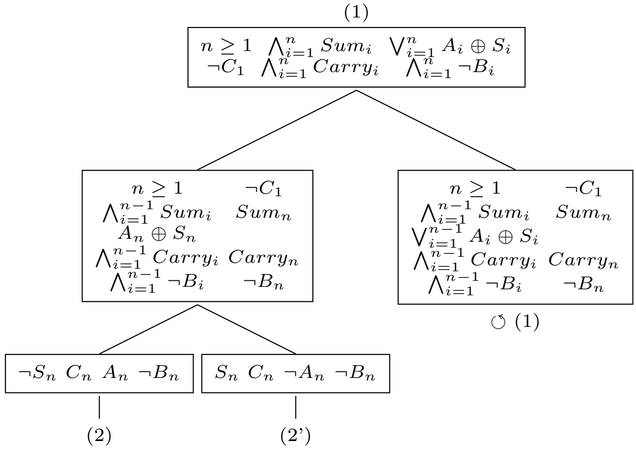
We consider an n-bits adder circuit: such a circuit is the composition of n 1-bit adders. The i^{th} bit of each operand is written A_i (resp. B_i). S_i is the i^{th} bit of the result, C_{i+1} is carried over to the next bit and $C_1 = 0$. We set the notations $Sum_i \overset{\text{def}}{=} S_i \Leftrightarrow (A_i \oplus B_i) \oplus C_i$ and $Carry_i \overset{\text{def}}{=} C_{i+1} \Leftrightarrow (A_i \wedge B_i) \vee (C_i \wedge A_i) \vee (C_i \wedge B_i)$ where $\oplus$ is the exclusive or. Then $Adder \overset{\text{def}}{=} \bigwedge_{i=1}^{n} Sum_i \wedge \bigwedge_{i=1}^{n} Carry_i \wedge \neg C_1$ with the constraint $n \geq 1$ schematises the adder circuit.

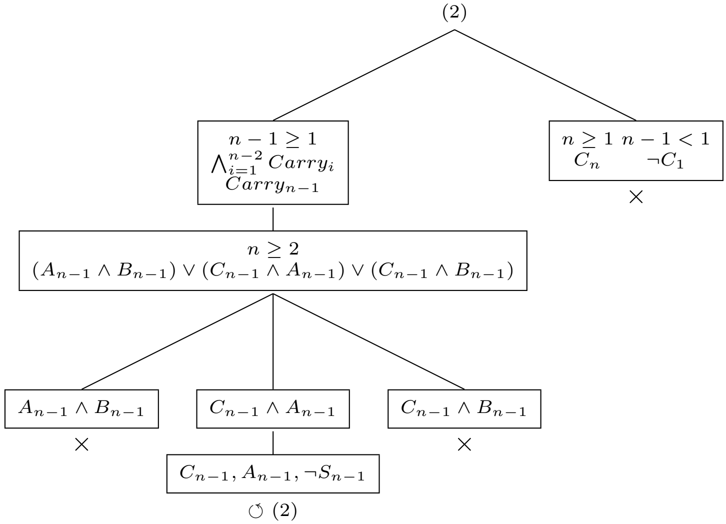
We aim at proving that $A + 0 = A$. A SAT-solver can easily refute this formula for a fixed n (say $n = 10$). We prove it for all $n \in \mathbb{N}$. This simple example has been chosen for the sake of readability and conciseness, notice that commutativity or associativity of the n-bits adder could have been proven too.

We express the fact that the second operand is null: $\bigwedge_{i=1}^{n} \neg B_i$, and the fact that the result equals the first operand: $\bigwedge_{i=1}^{n} A_i \Leftrightarrow S_i$, which gives $\bigvee_{i=1}^{n} A_i \oplus S_i$

by refutation. So we want to prove that $\mathit{Adder} \wedge \bigwedge_{i=1}^{n} \neg B_i \wedge \bigvee_{i=1}^{n} A_i \oplus S_i$ is unsatisfiable. Notice that this schema is regular.

We sketch the corresponding tableau, using the conventions that closed leaves (resp. blocked leaves looping on α) are marked by × (resp. $\circlearrowleft(\alpha)$). Sequences of propositional (resp. iteration) extension rules are not detailed and represented as thin (resp. thick) lines. Only new (w.r.t. the previous block) formulae are presented in the blocks.


(1)
$n \geq 1$ $\bigwedge_{i=1}^{n} Sum_i$ $\bigvee_{i=1}^{n} A_i \oplus S_i$
$\neg C_1$ $\bigwedge_{i=1}^{n} Carry_i$ $\bigwedge_{i=1}^{n} \neg B_i$
$n \geq 1$ $\neg C_1$
$\bigwedge_{i=1}^{n-1} Sum_i$ Sum_n
$A_n \oplus S_n$
$\bigwedge_{i=1}^{n-1} Carry_i$ $Carry_n$
$\bigwedge_{i=1}^{n-1} \neg B_i$ $\neg B_n$
$n \geq 1$ $\neg C_1$
$\bigwedge_{i=1}^{n-1} Sum_i$ Sum_n
$\bigvee_{i=1}^{n-1} A_i \oplus S_i$
$\bigwedge_{i=1}^{n-1} Carry_i$ $Carry_n$
$\bigwedge_{i=1}^{n-1} \neg B_i$ $\neg B_n$
$\circlearrowleft$ (1)
$\neg S_n$ C_n A_n $\neg B_n$
S_n C_n $\neg A_n$ $\neg B_n$
(2)
(2')



$(2')$ is very similar to (2).

Explanations. The first big step decomposes all the iterations. The branching is due to $\bigvee_{i=1}^{n} A_i \oplus S_i$: first we have $A_n \oplus S_n$, then $\bigvee_{i=1}^{n-1} A_i \oplus S_i$. The right branch loops after a few steps as all iterated conjunctions $\bigwedge_{i=1}^{n} \dots$ contain $\bigwedge_{i=1}^{n-1} \dots$

The left one is extended by propositional rules (the reader can easily check that Sum_n, $Carry_n$, $A_n \oplus S_n$ and $\neg B_n$ indeed lead to the presented branches).

In (2) we start by decomposing all iterations a second time. Similarly to (1) there is a branching due to $\bigvee_{i=1}^{n-1} A_i \oplus S_i$. We do not represent it as the second branch loops similarly. So we consider only the case where we have $A_{n-2} \oplus S_{n-2}$. Iterations are aligned on $[1..n-1]$ so they all introduce the same constraints i.e. either $n-1 \geq 1$ (first branch) or $n-1 < 1$ (second branch). In the second case, the introduced constraint implies that $n = 1$, thus $C_n = C_1$ which closes the branch. In the first case we decompose $Carry_{n-1}$ and consider the various cases. Two of them are trivially discarded as they imply B_{n-1}, whereas we have $\neg B_i$ for all $i \in [1..n]$. It only remains one case which is easily seen to loop on (2).

7 Conclusion

We presented the first (to the best of our knowledge) calculus for reasoning on iterated schemata of propositional formulae. We proved that this calculus is sound and complete (w.r.t. satisfiability). We identified a (reasonably expressive) class of schemata for which it is a decision procedure.

In our opinion, this work opens a new research area with many lines of future work. First an implementation is of course mandatory. Then the most natural follow-up is to extend STAB to first-order logic. The obtained language would allow one to denote mathematical proofs using induction on natural numbers, without having to use more expressive languages for which no proof procedures are available. It would also be useful to identify other syntactic classes of propositional schemata for which STAB terminates (as in Section 5, but allowing, for instance, nesting of iterations). Such classes could be obtained by restricting the form of the indices and/or the iteration patterns. Defining more powerful cases of the looping rule could be useful to this purpose. Finally schemata naturally arise in many symbolic computation procedures, particularly in programming and symbolic computation. We believe that the ideas and techniques introduced in the present paper could be reused in other domains, thus paving the ground for a general framework for iteration schemata.

Acknowledgements. We thank the three anonymous referees for their thorough reviews. Their remarks and suggestions have greatly contributed to improve the quality of the paper.

References

1. Corcoran, J.: Schemata: the concept of schema in the history of logic. The Bulletin of Symbolic Logic 12(2), 219–240 (2006)
2. Kneale, W., Kneale, M.: The development of logic. Clarendon Press, Oxford University Press (1986)
3. Baaz, M., Zach, R.: Short proofs of tautologies using the schema of equivalence. In: Meinke, K., Börger, E., Gurevich, Y. (eds.) CSL 1993. LNCS, vol. 832, pp. 33–35. Springer, Heidelberg (1994)

4. Parikh, R.J.: Some results on the length of proofs. Transactions of the American Mathematical Society 177, 29–36 (1973)
5. Baaz, M.: Note on the generalization of calculations. Theoretical Computer Science 224, 3–11 (1999)
6. Krajíček, J., Pudlák, P.: The number of proof lines and the size of proofs in first order logic. Archive for Mathematical Logic 27, 69–84 (1988)
7. Orevkov, V.P.: Proof schemata in Hilbert-type axiomatic theories. Journal of Mathematical Sciences 55(2), 1610–1620 (1991)
8. Wos, L., Overbeek, R., Lusk, E., Boyle, J.: Automated Reasoning, Introduction and Applications, 2nd edn. McGraw-Hill, New York (1992)
9. Barendregt, H., Wiedijk, F.: The challenge of computer mathematics. Philosophical Transactions of the Royal Society A 363, 2351–2375 (2005)
10. Wos, L.: Automated Reasoning: 33 Basic Research Problems. Prentice-Hall, Englewood Cliffs (1988)
11. Hermann, M., Galbavý, R.: Unification of Infinite Sets of Terms schematized by Primal Grammars. Theoretical Computer Science 176(1–2), 111–158 (1997)
12. Chen, H., Hsiang, J., Kong, H.: On finite representations of infinite sequences of terms. In: Okada, M., Kaplan, S. (eds.) CTRS 1990. LNCS, vol. 516, pp. 100–114. Springer, Heidelberg (1991)
13. Comon, H.: On unification of terms with integer exponents. Mathematical System Theory 28, 67–88 (1995)
14. Boyer, R.S., Moore, J.S.: A computational logic. Academic Press, London (1979)
15. Bouhoula, A., Kounalis, E., Rusinowitch, M.: SPIKE, an automatic theorem prover. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624, pp. 460–462. Springer, Heidelberg (1992)
16. Comon, H.: Inductionless induction. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 913–962. North-Holland, Amsterdam (2001)
17. Bundy, A., van Harmelen, F., Horn, C., Smaill, A.: The Oyster-Clam system. In: Stickel, M.E. (ed.) CADE 1990. LNCS, vol. 449, pp. 647–648. Springer, Heidelberg (1990)
18. Stratulat, S.: Automatic 'Descente Infinie' Induction Reasoning. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS, vol. 3702, pp. 262–276. Springer, Heidelberg (2005)
19. Wirth, C.P., Becker, K.: Abstract notions and inference systems for proofs by mathematical induction. In: Lindenstrauss, N., Dershowitz, N. (eds.) CTRS 1994. LNCS, vol. 968, pp. 353–373. Springer, Heidelberg (1995)
20. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability modulo theories. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, February 2009, vol. 185, pp. 825–885. IOS Press, Amsterdam (2009)
21. Armando, A., Ranise, S., Rusinowitch, M.: A rewriting approach to satisfiability procedures. Information and Computation 183(2), 140–164 (2003)
22. Smullyan, R.M.: First-Order Logic. Springer, Heidelberg (1968)
23. Fitting, M.: First-Order Logic and Automated Theorem Proving. Texts and Monographs in Computer Science. Springer, Heidelberg (1990)
24. Cooper, D.: Theorem proving in arithmetic without multiplication. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 7, pp. 91–99. Edinburgh University Press (1972)

Tableaux and Model Checking for Memory Logics

Carlos Areces[1], Diego Figueira[2], Daniel Gorín[3], and Sergio Mera[3,⋆]

[1] INRIA Nancy Grand Est, Nancy, France
areces@loria.fr
[2] INRIA Saclay, ENS Cachan, LSV, France
figueira@lsv.ens-cachan.fr
[3] Departamento de Computación, UBA, Argentina
{dgorin,smera}@dc.uba.ar

Abstract. Memory logics are modal logics whose semantics is specified in terms of relational models enriched with additional *data structure* to represent *memory*. The logical language is then extended with a collection of operations to access and modify the data structure. In this paper we study their satisfiability and the model checking problems.

We first give sound and complete tableaux calculi for the memory logic $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ (the basic modal language extended with the operator $\textcircled{r}$ used to memorize a state, the operator $\textcircled{e}$ used to wipe out the memory, and the operator $\textcircled{k}$ used to check if the current point of evaluation is memorized) and some of its sublanguages. As the satisfiability problem of $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ is undecidable, the tableau calculus we present is non terminating. Hence, we furthermore study a variation that ensures termination, at the expense of completeness, and we use model checking to ensure soundness. Secondly, we show that the model checking problem is PSpace-complete.

1 Memory Logics

In a number of recent papers [1,2,3,4] we have investigated a family of logics that we call *memory logics*. These logics are related to both modal logics [5,6] and hybrid logics [7], as well as other logics that intend to add some notion of state to models [8,9,10,11,12].

Intuitively, memory logics are modal logics whose semantics is specified in terms of first-order relational models enriched with additional *data structure* to represent *memory*. The logical language is then extended with a collection of operations to access and modify the data structure.

Formally, let $\mathcal{M} = \langle W, (R_r)_{r \in \mathsf{Rel}}, V \rangle$ be a relational structure where W is a non empty domain; for each relation symbol r, R_r is a binary relation over W; and $V : \mathsf{Prop} \to 2^W$ is a valuation function that assigns subsets of W to propositional symbols in Prop. We can extend this structure with a set $S \subseteq W$ which can be interpreted as a set of states that are 'known' to us, and which

[⋆] S. Mera is partially supported by a grant of Fundación YPF.

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 47–61, 2009.

represent the current 'memory' of the model. Even in this simple setting we can define the following operators:

$$\begin{array}{lll}
\langle W, (R_r)_{r \in \mathsf{Rel}}, V, S\rangle, w \models \textcircled{r}\varphi & \text{iff} & \langle W, (R_r)_{r \in \mathsf{Rel}}, V, S \cup \{w\}\rangle, w \models \varphi, \\
\langle W, (R_r)_{r \in \mathsf{Rel}}, V, S\rangle, w \models \textcircled{e}\varphi & \text{iff} & \langle W, (R_r)_{r \in \mathsf{Rel}}, V, \emptyset\rangle, w \models \varphi, \\
\langle W, (R_r)_{r \in \mathsf{Rel}}, V, S\rangle, w \models \textcircled{k} & \text{iff} & w \in S.
\end{array}$$

As it is clear from the definition above, the 'remember' operator $\textcircled{r}$ (a unary modality) just marks the current state as being 'known' or 'already visited', by storing it in our 'memory' S. The 'erase' operator $\textcircled{e}$ (also unary) wipes out the memory. These are the operators we use to update the memory. On the other hand, the zero-ary operator $\textcircled{k}$ (for 'known') queries S to check if the current state has already been visited. Notice that the extension of S is dynamic and it can vary during the evaluation of a formula.

Our original motivation to investigate memory logics was mainly theoretical: we were looking for a modal language that included some kind of binding mechanism (notice that $\textcircled{r}$ effectively binds instances of $\textcircled{k}$ appearing in its scope), but with a decidable satisfiability problem. The memory logic $\mathcal{ML}(\textcircled{k}, \textcircled{r})$ (i.e., the basic modal language extended with only the $\textcircled{k}$ and $\textcircled{r}$ operators) was introduced as a weakening of the operator $\downarrow$ from the hybrid logic $\mathcal{HL}(\downarrow)$ (i.e., the basic modal language extended with nominals and the $\downarrow$ binder [7]) known to be undecidable. But, as we have shown in [2,3] , even though the language is strictly less expressive than $\mathcal{HL}(\downarrow)$, its satisfiability problem is still undecidable.

While working with the memory operators we realised that they provide an interesting perspective on modalities and their interaction with models: they are examples of operators that *modify* the model during evaluation, and in that sense they are truly *dynamic*. They are examples of logical languages that could both *check* conditions on the model, and *modify* the model accordingly. For example, while evaluating the formula $\textcircled{r}\psi$ in a model $\mathcal{M}$, the $\textcircled{r}$ operator transforms $\mathcal{M}$ into a new model $\mathcal{M}'$ (by adding the current point of evaluation to the memory), and ψ is then evaluated in $\mathcal{M}'$. We could imagine other operators that modify $\mathcal{M}$ in different ways: add states, change the valuation, modify accessibility relations, etc. By investigating memory logics we want to understand the basic properties of such languages. From this perspective, memory logics would be related to other well-known logics. One example are dynamic epistemic logics [8], which are languages used to model the evolution of the knowlege of epistemic agents via *updates* to the model representing their epistemic state. Other approach comes from temporal logics with explicit global clocks (for example, the logic XCTL [9]), in which these clocks are accessed and controlled through logical operators. We believe that by studying the memory logics family we will better understand some of the basic notions and intuitions that all these languages have in common.

In this article we investigate computational aspects of two classical reasoning tasks for memory logics. In Section 2 we develop sound and complete tableau calculi. As the satisfiability problem for $\mathcal{ML}(\textcircled{k}, \textcircled{r})$ (and hence also the one for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e}))$ is undecidable – and given that the calculi are sound and complete – the tableaux obtained by the application of the rules we provide might be infinite. In Section 3 we restrict these calculi so that they always produce finite tableaux, but at the expense of sacrificing completeness. For this restriction to work, we will need to perform model-checking (over an induced model) and in Section 4 we investigate the complexity of this task. Because $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ is a fragment of first-order logic, we know that the problem is in PSpace [13]. We will show that it actually is PSpace-complete.

2 Complete and Sound Tableau Calculi

In this section we will introduce a tableau calculus for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ (as we will explain below, we will actually propose calculi over two particularly interesting classes of models, and discuss also calculi for some sublogics). To make the paper self-contained, we start by introducing some notation and basic notions.

Definition 1 (Syntax). *Let* $\mathsf{Prop} = \{p_1, p_2, \dots\}$ *(the* propositional symbols*) and* $\mathsf{Rel} = \{r_1, r_2, \dots\}$ *(the* relational symbols*) be disjoint, countable infinite sets.* Forms*, the set of formulas of* $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ *over signature* $\langle \mathsf{Prop}, \mathsf{Rel} \rangle$*, is defined as:*

$$\mathsf{Forms} ::= p \mid \neg p \mid \textcircled{k} \mid \neg \textcircled{k} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \langle r \rangle \varphi \mid [r]\varphi \mid \textcircled{r}\varphi \mid \textcircled{e}\varphi,$$

where $p \in \mathsf{Prop}$*,* $r \in \mathsf{Rel}$ *and* $\varphi, \varphi_1, \varphi_2 \in \mathsf{Forms}$*.*

Given $\varphi \in \mathsf{Forms}$ *we will write* $\overline{\varphi}$ *for the formula obtained by computing the* negated normal form *of the negation of* φ*:*

$$\begin{array}{lllll} \overline{p} = \neg p & \overline{\textcircled{k}} = \neg \textcircled{k} & \overline{\varphi_1 \wedge \varphi_2} = \overline{\varphi_1} \vee \overline{\varphi_2} & \overline{\langle r \rangle \varphi} = [r]\overline{\varphi} & \overline{\textcircled{r}\varphi} = \textcircled{r}\overline{\varphi} \\ \overline{\neg p} = p & \overline{\neg \textcircled{k}} = \textcircled{k} & \overline{\varphi_1 \vee \varphi_2} = \overline{\varphi_1} \wedge \overline{\varphi_2} & \overline{[r]\varphi} = \langle r \rangle \overline{\varphi} & \overline{\textcircled{e}\varphi} = \textcircled{e}\overline{\varphi} \end{array}$$

Sublogics of $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ are obtained by forbidding the use of certain operators. For example, $\mathcal{ML}(\textcircled{k}, \textcircled{r})$ is the logic obtained by restricting to formulas in Forms not containing $\textcircled{e}$.

Definition 2 (Semantics). *Given a signature* $\mathcal{S} = \langle \mathsf{Prop}, \mathsf{Rel} \rangle$*, a* model *for* $\mathcal{S}$ *is a tuple* $\langle W, (R_r)_{r \in \mathsf{Rel}}, V, S \rangle$*, satisfying the following conditions: (i)* $W \neq \emptyset$*; (ii) each* R_r *is a binary relation on* W*; (iii)* $V : \mathsf{Prop} \to 2^W$ *is a labeling function; and (iv)* $S \subseteq W$*.*

For any model $\mathcal{M} = \langle W, (R_r)_{r \in \mathsf{Rel}}, V, S \rangle$*, we will denote with* $\mathcal{M}[w_1, \dots, w_n]$ *and* $\mathcal{M}_\emptyset$ *the models* $\langle W, (R_r)_{r \in \mathsf{Rel}}, V, S \cup \{w_1, \dots, w_n\} \rangle$ *and* $\langle W, (R_r)_{r \in \mathsf{Rel}}, V, \emptyset \rangle$ *respectively.*

Given the model $\mathcal{M} = \langle W, (R_r)_{r \in \mathsf{Rel}}, V, S \rangle$ *and* $w \in W$*, the semantics for the different operators is defined as follows:*

$$\begin{array}{lcl}
\mathcal{M}, w \models p & \iff & w \in V(p), \quad p \in \mathsf{Prop} \\
\mathcal{M}, w \models \neg p & \iff & w \notin V(p), \quad p \in \mathsf{Prop} \\
\mathcal{M}, w \models \varphi \wedge \psi & \iff & \mathcal{M}, w \models \varphi \textit{ and } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \varphi \vee \psi & \iff & \mathcal{M}, w \models \varphi \textit{ or } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \langle r \rangle \varphi & \iff & \textit{there is } w' \textit{ such that } R_r(w, w') \textit{ and } \mathcal{M}, w' \models \varphi \\
\mathcal{M}, w \models [r] \varphi & \iff & \textit{for all } w' \textit{ such that } R_r(w, w'),\ \mathcal{M}, w' \models \varphi \\
\mathcal{M}, w \models \textcircled{\texttt{r}} \varphi & \iff & \mathcal{M}[w], w \models \varphi \\
\mathcal{M}, w \models \textcircled{\texttt{e}} \varphi & \iff & \mathcal{M}_{\emptyset}, w \models \varphi \\
\mathcal{M}, w \models \textcircled{\texttt{k}} & \iff & w \in S \\
\mathcal{M}, w \models \neg \textcircled{\texttt{k}} & \iff & w \notin S.
\end{array}$$

Definition 3 (Satisfiability and Validity). *Let $\mathcal{C}$ be a class of models. We say that φ is* satisfiable in $\mathcal{C}$ *if there is a model $\mathcal{M} \in \mathcal{C}$ and a state w in the domain of $\mathcal{M}$ such that $\mathcal{M}, w \models \varphi$. We say that φ is* valid in $\mathcal{C}$ *if $\overline{\varphi}$ is not satisfiable in $\mathcal{C}$.*

We will be mainly interested in using tableaux to characterize the set of valid formulas over $\mathcal{C}_{\text{all}}$, the class of all models. But observe that to express several structural properties of interest, it is natural to start with a model with no previously remembered states.

For example, $\langle W, (R_r)_{r \in \mathsf{Rel}}, \emptyset \rangle, w \models \textcircled{\texttt{r}}\langle r \rangle \textcircled{\texttt{k}}$ if and only if $R(w, w)$. That is, satisfiability of $\textcircled{\texttt{r}}\langle r \rangle \textcircled{\texttt{k}}$ at w characterizes reflexivity of w whenever the initial memory is empty. When the $\textcircled{\texttt{e}}$ operator is in the language, we can actually use the formula $\textcircled{\texttt{e}}\textcircled{\texttt{r}}\langle r \rangle \textcircled{\texttt{k}}$ instead, which ensures that S is empty before evaluating $\textcircled{\texttt{r}}\langle r \rangle \textcircled{\texttt{k}}$. That is, if $\mathcal{C}_{\emptyset}$ is the class $\{\mathcal{M} \mid \mathcal{M} = \langle W, (R_r)_{r \in \mathsf{Rel}}, V, \emptyset \rangle\}$ of models with an empty memory, then φ is valid in $\mathcal{C}_{\emptyset}$ iff $\textcircled{\texttt{e}}\varphi$ is valid in $\mathcal{C}_{\text{all}}$. Or in other words, we can use $\textcircled{\texttt{e}}$ to 'internalize' the class $\mathcal{C}_{\emptyset}$.

Because we will discuss not only the full language $\mathcal{ML}(\textcircled{\texttt{k}}, \textcircled{\texttt{r}}, \textcircled{\texttt{e}})$, but also some of its sublanguages, we'll set up the tableau calculi so that they can be used for satisfiability for both $\mathcal{C}_{\text{all}}$ and $\mathcal{C}_{\emptyset}$.

In Figure 1 we present the rules for a prefixed tableau calculus for the logic $\mathcal{ML}(\textcircled{\texttt{k}}, \textcircled{\texttt{r}}, \textcircled{\texttt{e}})$. Prefixed tableaux for hybrid logics were investigated by Blackburn and Bolander in [14]. The general approach and, in particular, the termination argument used in Section 3 are inspired by this paper.

A tableau in the calculus presented in Figure 1 is simply a wellfounded, finitely branching tree in which edges represent applications of tableau rules in the usual way and each node is labeled by an accessibility, equality or prefixed formula.

Definition 4 (Prefixed, accessibility and equality formulas). *Let $W = \{w_1, w_2, \ldots\}$ be an infinite, enumerable set of labels. Then $\langle w, A \rangle^C{:}\varphi$ is a* prefixed formula*, where $\varphi \in \mathcal{ML}(\textcircled{\texttt{k}}, \textcircled{\texttt{r}}, \textcircled{\texttt{e}})$, $C \in \{\mathcal{C}_{\text{all}}, \mathcal{C}_{\emptyset}\}$, $w \in W$ and A is a finite subset of W. $R_r(w, w')$ is an* accessibility formula *for $r \in \mathsf{Rel}$, and $w, w' \in W$. $w \approx w'$ is an* equality formula *for $w, w' \in W$.*

Intuitively, in the prefix $\langle w, A \rangle^C$, w is the label of the state where the formula holds, C is the class of models we are working with ($\mathcal{C}_{\text{all}}$ or $\mathcal{C}_{\emptyset}$), and A is a set

$$(\wedge)\ \frac{\langle w,A\rangle^{C}{:}\varphi\wedge\psi}{\begin{array}{c}\langle w,A\rangle^{C}{:}\varphi\\ \langle w,A\rangle^{C}{:}\psi\end{array}}\qquad\qquad (\vee)\ \frac{\langle w,A\rangle^{C}{:}\varphi\vee\psi}{\langle w,A\rangle^{C}{:}\varphi \mid \langle w,A\rangle^{C}{:}\psi}$$

$$(\langle r\rangle)\ \frac{\langle w,A\rangle^{C}{:}\langle r\rangle\varphi}{\begin{array}{c}R_r(w,w')\\ \langle w',A\rangle^{C}{:}\varphi\end{array}}\ \dagger\qquad\qquad ([r])\ \frac{\begin{array}{c}\langle w,A\rangle^{C}{:}[r]\varphi\\ R_r(w,w')\end{array}}{\langle w',A\rangle^{C}{:}\varphi}$$

$$(\neg\textcircled{k})\ \frac{\langle w,A\rangle^{C}{:}\neg\textcircled{k}}{\langle w,\emptyset\rangle^{C}{:}\neg\textcircled{k}}\qquad\qquad (\textcircled{k})\ \frac{\langle w,\{v_1,\ldots v_k\}\rangle^{C}{:}\textcircled{k}}{w\approx v_1 \mid \cdots \mid w\approx v_k \mid \langle w,\emptyset\rangle^{C}{:}\textcircled{k}}$$

$$(\textcircled{e})\ \frac{\langle w,A\rangle^{C}{:}\textcircled{e}\varphi}{\langle w,\emptyset\rangle^{C_\emptyset}{:}\varphi}\qquad\qquad (\textcircled{r})\ \frac{\langle w,A\rangle^{C}{:}\textcircled{r}\varphi}{\langle w,A\cup\{w\}\rangle^{C}{:}\varphi}$$

$$(\text{repl})\ \frac{\begin{array}{c}\langle w,A\rangle^{C}{:}\varphi\\ w\approx^{*} w'\end{array}}{\langle w',A[w\mapsto w']\rangle^{C}{:}\varphi}\ \ddagger$$

Clash Rules:

$$(\bot_p)\ \frac{\begin{array}{c}\langle w,A\rangle^{C_1}{:}p\\ \langle w,B\rangle^{C_2}{:}\neg p\end{array}}{\bot}\qquad\qquad (\bot_{\textcircled{k}})\ \frac{\begin{array}{c}\langle w,\emptyset\rangle^{C}{:}\textcircled{k}\\ \langle w,\emptyset\rangle^{C}{:}\neg\textcircled{k}\end{array}}{\bot}$$

$$(\bot_{\neg\textcircled{k}})\ \frac{\langle w,\{w\}\cup A\rangle^{C}{:}\neg\textcircled{k}}{\bot}\qquad\qquad (\bot_{\emptyset})\ \frac{\langle w,\emptyset\rangle^{\mathcal{C}_\emptyset}{:}\textcircled{k}}{\bot}$$

Key:

† w' is fresh.

‡ $a\approx^{*} b$ iff (a,b) occurs in the reflexive, symmetric and transitive closure of the relation $\{(w,w') \mid w\approx w'$appears in the current branch$\}$. $A[w\mapsto w'] = A$ if $w\notin A$, and $(A-\{w\})\cup\{w'\}$ otherwise.

- C, C_1, C_2 are either $\mathcal{C}_{\text{all}}$ or $\mathcal{C}_\emptyset$.
- A, B are arbitrary finite set of labels.

Fig. 1. Tableau rules

of states that were explicitly remembered by evaluating a $\textcircled{r}$ operator in the current branch. Since every prefixed formula is derived in finitely many steps, A will always be a finite set. In the rest of this article we will refer to a tableau that uses the rules presented in Figure 1 as a tableau for $\mathcal{ML}(\textcircled{k},\textcircled{r},\textcircled{e})$. The intended interpretation of $R_r(w,w')$ is that the state denoted by w' is accessible from the state denoted by w by the interpretation of relation symbol r. Finally, the intended interpretation of an equality formula $w \approx w'$ is that w and w' label the same state in a given branch.

We will use the term *formula* to denote either a formula of $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$, a prefixed formula, an accessibility formula, or an equality formula.

The rules are presented in the standard format: each rule has a name on the left and is divided in an upper (the antecedent) and lower (the consequent) part. Whenever there are formulas in a branch that match the antecedent, the rule can be applied following the constraints specified for each rule. If the rule is applied, the formulas of the consequent are added to the same branch, except in the case of $(\vee)$ and $(\textcircled{k})$, where several different branches are created.

The rules $(\langle r \rangle)$, $(\neg\textcircled{k})$, $(\textcircled{k})$, $(\textcircled{e})$, $(\textcircled{r})$ and (repl) are called "prefix generating rules", since if a prefix is new to a branch, it must be introduced by one of these rules. We impose two general constraints on the construction of a tableau:

- A prefix generating rule is never applied twice to a formula on a given branch.
- A formula is never added to a tableau branch where it already occurs.

A *saturated tableau* is a tableau in which no more rules can be applied that satisfy the constraints. A *saturated branch* is a branch of a saturated tableau. A branch of a tableau is called *closed* if it contains $\bot$. Otherwise the branch is called *open*. A *closed tableau* is one in which all branches are closed, and an *open tableau* is one in which at least one branch is open.

$(\wedge)$, $(\vee)$, $(\langle r \rangle)$ and $([r])$ are classical rules of the basic modal logic tableau calculus. The remaining ones are particular to memory logics. Rule $(\neg\textcircled{k})$ specifies that at a label where A denotes the set of states that were explicitly remembered, if the state w is not in the memory then $w \notin A$ and (in particular) w still is not memorized at the label with $A = \emptyset$. Rule $(\textcircled{k})$ specifies that if w is in the memory, then either it is one of the explicitly remembered states, or it is in the initial memory, in which case $\textcircled{k}$ holds even with no explicitly remembered states. Notice that the last branch of the application of this rule can be immediately closed in the case where $C = C_\emptyset$, due to the rule $(\bot_\emptyset)$. Rule $(\textcircled{e})$ wipes out the explicitly remembered states and evaluates the satisfiability of the formula in a model with no initial memory. Observe that the presence of the $\textcircled{e}$ modality may force the calculus to switch from the evaluation over C_{all} to that over $C_\emptyset$.

We will also consider variations and subsystems of the calculus of Figure 1 where only a subset of the rules are allowed, or additional constraints on the rules are imposed (for example, to ensure termination). In such subsystems, a tableau is of course simply a tableau in which only the rules in the subset can be applied, considering the additional constraints.

We call a tableau calculus $\mathcal{T}$ *sound* for a language $\mathcal{L}$ respect to a class of models $\mathcal{C}$ if whenever $\varphi \in \mathcal{L}$ is $\mathcal{C}$-satisfiable, then every saturated tableau T with root φ has an open branch. We say that it is *complete* if whenever $\varphi \in \mathcal{L}$ is not $\mathcal{C}$-satisfiable, then every saturated tableau T with root φ is closed.

Soundness of the calculus of Figure 1 follows from a simple inspection of the rules. We devote the rest of this section to prove completeness. As usual, we will show that given an open and saturated branch Γ, we can define a model $\mathcal{M}^\Gamma$

that satisfies all the formulas that occur in the branch. To define the domain of $\mathcal{M}^\Gamma$ we first need the following definition.

Definition 5 **(Eq_Γ).** *Let Γ be an open and saturated branch of a tableau for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$. Eq_Γ is the smallest equivalence relation extending $\{(w, w') \mid (w \approx w') \in \Gamma\}$.*

Definition 6 **($\mathcal{M}^\Gamma$).** *Let Γ be an open and saturated branch of a tableau for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$. Define the induced model $\mathcal{M}^\Gamma = \langle W_\Gamma, (R_{r\Gamma})_{r \in \mathsf{Rel}}, V_\Gamma, S_\Gamma \rangle$ as:*

$$\begin{array}{ll} W_\Gamma &= \{w \mid w \textit{ occurs in } \Gamma\}/\mathrm{Eq}_\Gamma \\ R_{r\Gamma} &= \{([w],[w']) \mid R_r(w,w') \in \Gamma\} \\ V_\Gamma(p) &= \{[w] \mid \langle w, A\rangle^C{:}p \in \Gamma, \textit{ for any } A \textit{ and } C\} \\ S_\Gamma &= \{[w] \mid \langle w, \emptyset\rangle^{\mathcal{C}_{all}}{:}\textcircled{k} \in \Gamma\}, \end{array}$$

where $[w]$ is the equivalence class of w in Eq_Γ.

Lemma 1. *Let $\mathcal{M}^\Gamma = \langle W_\Gamma, (R_{r\Gamma})_{r \in \mathsf{Rel}}, V_\Gamma, S_\Gamma \rangle$ be the induced model for Γ, where Γ is an open and saturated branch of a tableau for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$.*

1. *$\langle w, \{v_1, \ldots v_k\}\rangle^{\mathcal{C}_{all}}{:}\varphi \in \Gamma$ implies $\mathcal{M}^\Gamma[[v_1], \ldots, [v_k]], [w] \models \varphi$.*
2. *$\langle w, \{v_1, \ldots v_k\}\rangle^{\mathcal{C}_\emptyset}{:}\varphi \in \Gamma$ implies $\mathcal{M}^\Gamma_\emptyset[[v_1], \ldots, [v_k]], [w] \models \varphi$.*

Proof. We proceed by induction on φ.

Case $\varphi := p$. If $\langle w, \{v_1, \ldots, v_k\}\rangle^{\mathcal{C}_{\mathrm{all}}}{:}p \in \Gamma$, then $[w] \in V_\Gamma(p)$ and, therefore, $\mathcal{M}^\Gamma[[v_1], \ldots, [v_k]], [w], \models p$. The case for $\mathcal{C}_\emptyset$ is analogous.

Case $\varphi := \neg p$. Suppose $\langle w, \{v_1, \ldots, v_k\}\rangle^C{:}\neg p \in \Gamma$. If $\mathcal{M}^\Gamma[[v_1], \ldots, [v_k]], [w] \models p$, it means that $[w] \in V_\Gamma(p)$ and hence $\langle w, A\rangle^C{:}p \in \Gamma$ (for some A and C), but in that case rule $(\bot_p)$ applies and the branch would be closed. Again, the case for $\mathcal{C}_\emptyset$ is analogous.

Case $\varphi := \textcircled{k}$. We consider all the different possibilities:

1. If $\langle w, \emptyset\rangle^{\mathcal{C}_{\mathrm{all}}}{:}\textcircled{k} \in \Gamma$, then $[w] \in S_\Gamma$ and, therefore, $\mathcal{M}^\Gamma, [w] \models \textcircled{k}$.
2. If $\langle w, \emptyset\rangle^{\mathcal{C}_\emptyset}{:}\textcircled{k} \in \Gamma$, then, by the $(\bot_\emptyset)$ rule, $\bot \in \Gamma$ which would contradict the hypothesis that Γ is an open branch.
3. If $\langle w, \{v_1, \ldots v_k\}\rangle^{\mathcal{C}_{\mathrm{all}}}{:}\textcircled{k} \in \Gamma$, with $k > 0$ then some consequent of the $(\textcircled{k})$ rule must occur in Γ too. If $\langle w, \emptyset\rangle^{\mathcal{C}_{\mathrm{all}}}{:}\textcircled{k} \in \Gamma$ then we are done. So let us assume that, on the contrary, $w \approx v_i \in \Gamma$ for some $i \in \{1, \ldots k\}$. This implies that $[w] = [v_i]$, but since $v_i \in \{v_1, \ldots v_k\}$, we conclude that $\mathcal{M}^\Gamma[[v_1], \ldots, [v_k]], [w] \models \textcircled{k}$.
4. The case when $\langle w, \{v_1, \ldots v_k\}\rangle^{\mathcal{C}_\emptyset}{:}\textcircled{k} \in \Gamma$, with $k > 0$ is analogous.

Case $\varphi := \neg\textcircled{k}$. We consider, again, all the distinct cases:

1. Let $\langle w, \emptyset\rangle^{\mathcal{C}_{\mathrm{all}}}{:}\neg\textcircled{k} \in \Gamma$ and let us assume, for the sake of contradiction, that $\mathcal{M}^\Gamma, [w] \models \textcircled{k}$. This means that $[w] \in S_\Gamma$ and, therefore, $\langle w', \emptyset\rangle^{\mathcal{C}_{\mathrm{all}}}{:}\textcircled{k} \in \Gamma$, where $[w] = [w']$. Since Γ is saturated, by the (repl) rule we know that $\langle w, \emptyset\rangle^{\mathcal{C}_{\mathrm{all}}}{:}\textcircled{k} \in \Gamma$. But then rule $(\bot_{\textcircled{k}})$ applies and $\bot \in \Gamma$ which makes Γ a closed branch.
2. Suppose $\langle w, \emptyset\rangle^{\mathcal{C}_\emptyset}{:}\neg\textcircled{k} \in \Gamma$. It is always the case that $\mathcal{M}^\Gamma_\emptyset, [w] \models \neg\textcircled{k}$.

3. Let $\langle w, \{v_1, \dots v_k\}\rangle^{\mathcal{C}_{\text{all}}}{:}\neg\textcircled{k} \in \Gamma$, with $k > 0$, and suppose, for the sake of contradiction, that $\mathcal{M}^{\Gamma}[[v_1], \dots, [v_k]], [w] \models \textcircled{k}$. This opens two possibilities. First, it could be the case that $[w] \in S_{\Gamma}$, but that would mean that $\langle w, \emptyset\rangle^{\mathcal{C}_{\text{all}}}{:}\textcircled{k} \in \Gamma$ and, because of the $(\neg\textcircled{k})$ rule, $\langle w, \emptyset\rangle^{\mathcal{C}_{\text{all}}}{:}\neg\textcircled{k} \in \Gamma$ and therefore we would have a clash by the $(\bot_{\textcircled{k}})$ rule.

   Alternatively, it could be the case that $[w] = [v_i]$ for some $i \in \{1, \dots k\}$. Since Γ is saturated, we conclude $\langle v_i, \{v_1, \dots v_k\}[w \mapsto v_i]\rangle^{\mathcal{C}_{\text{all}}}{:}\neg\textcircled{k} \in \Gamma$ using the (repl) rule. But observe that $v_i \in \{v_1, \dots v_k\}[w \mapsto v_i]$ from which rule $(\bot_{\neg\textcircled{k}})$ applies and leads to a contradiction.
4. If $\langle w, \{v_1, \dots v_k\}\rangle^{\mathcal{C}_{\emptyset}}{:}\neg\textcircled{k} \in \Gamma$, with $k > 0$, we can simply use the argument for the case $[w] = [v_i]$ just above.

Case $\varphi := \textcircled{r}\psi$. Suppose $\langle w, \{v_1, \dots v_k\}\rangle^{\mathcal{C}_{\text{all}}}{:}\textcircled{r}\psi \in \Gamma$. By rule $(\textcircled{r})$, we know $\langle w, \{v_1, \dots v_k, w\}\rangle^{\mathcal{C}_{\text{all}}}{:}\psi \in \Gamma$. By inductive hypothesis, $\mathcal{M}^{\Gamma}[[v_1], \dots [v_k], [w]], [w] \models \psi$, which implies $\mathcal{M}^{\Gamma}[[v_1], \dots [v_k]], [w] \models \textcircled{r}\psi$. $\mathcal{C}_{\emptyset}$ is analogous.

Case $\varphi := \textcircled{e}\psi$. If $\langle w, \{v_1, \dots v_k\}\rangle^{\mathcal{C}_{\text{all}}}{:}\textcircled{e}\psi \in \Gamma$ then, by rule $(\textcircled{e})$, $\langle w, \emptyset\rangle^{\mathcal{C}_{\emptyset}}{:}\psi \in \Gamma$ and, by inductive hypothesis, $\mathcal{M}^{\Gamma}_{\emptyset}, [w] \models \psi$. Therefore, it follows that $\mathcal{M}[[v_1], \dots [v_k]], [w] \models \textcircled{e}\psi$. The case for $\mathcal{C}_{\emptyset}$ is analogous.

The remaining boolean and modal cases are dealt with in the standard way.

Theorem 1. *The tableau calculus for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ is sound and complete for both the classes $\mathcal{C}_{\text{all}}$ and $\mathcal{C}_{\emptyset}$.*

More precisely, given $\varphi \in \mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$, φ is satisfiable iff any saturated tableau for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ with root $\langle w, \emptyset\rangle^{\mathcal{C}_{\text{all}}}{:}\varphi$ has an open branch. An equivalent result holds for the $\mathcal{C}_{\emptyset}$ class, starting with a tableau with root $\langle w, \emptyset\rangle^{\mathcal{C}_{\emptyset}}{:}\varphi$.

Proof. Soundness is trivial. Completeness is straightforward from Lemma 1: assume that a formula $\varphi \in \mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ is not satisfiable in the class C while there is a saturated tableau T with root $\langle w, \emptyset\rangle^{C}{:}\varphi$ and open branch Γ; $\mathcal{M}^{\Gamma}$ satisfies φ and is in the class C which contradicts the assumption.

It is also straightforward to see that if we drop the $(\textcircled{e})$ rule from the calculus, then we can prove soundness and completeness for formulas in $\mathcal{ML}(\textcircled{r}, \textcircled{k})$ (again with respect to both classes $\mathcal{C}_{\text{all}}$ and $\mathcal{C}_{\emptyset}$).

Theorem 2. *The tableau calculus of Figure 1 without the $(\textcircled{e})$ rule is sound and complete for $\mathcal{ML}(\textcircled{r}, \textcircled{k})$ for both the classes $\mathcal{C}_{\text{all}}$ and $\mathcal{C}_{\emptyset}$.*

3 Terminating Tableaux

In this section we will investigate some constraints that can be applied to the tableau rules for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ in order to ensure termination. The price we have to pay is that the resulting calculus is not complete any more. More precisely, it will be the case that some formula φ has a tableau with an open saturated branch Γ whose induced model $\mathcal{M}^{\Gamma}$ is not a model for φ. This means that we cannot claim satisfiability of the root formula every time we obtain a saturated open tableau. In these cases, we will use a model checking algorithm to verify whether $\mathcal{M}^{\Gamma}$ is effectively a model for φ. If the model checking succeeds we can then indeed answer SAT, and we will answer NOT-KNOWN otherwise.

We begin this section defining the restricted tableau rules, and proving a termination result. After this we will formalize the connection with model checking. In what follows, when a prefixed formula $\sigma{:}\varphi$ occurs in a tableau branch Γ we will write $\sigma{:}\varphi \in \Gamma$, and say that φ is true at σ on Γ or that σ makes φ true on Γ. Also, given a prefix $\sigma = \langle w, A\rangle^{C}$ we will define $\mathit{Label}(\sigma) = w$ and $\mathit{Set}(\sigma) = A$.

We will start by showing that by eliminating the (repl) rule one obtains a terminating calculus.

Definition 7. *Given a tableau branch Γ and a prefix σ, the set of true formulas at σ on Γ, written $T_{\Gamma}(\sigma)$, is defined as $T_{\Gamma}(\sigma) = \{\varphi \mid \sigma{:}\varphi \in \Gamma\}$.*

Notice that accessibility and equality formulas are not included in $T_{\Gamma}(\sigma)$.

Lemma 2 (Subformula Property). *Let T be a tableau with the prefixed formula $\sigma_0{:}\varphi_0$ as root. For any prefixed formula $\sigma{:}\varphi$ occurring on T, φ is a subformula of φ_0.*

Proof. This is easily seen by going through each of the tableau rules.

Lemma 3. *Let Γ be a branch of a tableau, and let σ be any prefix occurring on Γ. The set $T_{\Gamma}(\sigma)$ is finite.*

Proof. Let $\sigma_0{:}\varphi_0$ denote the first formula on Γ (i.e., the root of the tableau). From Lemma 2, we know that $T_{\Gamma}(\sigma) \subseteq \{\varphi \mid \varphi \text{ is a subformula of } \varphi_0\}$, and hence $T_{\Gamma}(\sigma)$ is finite.

Definition 8. *Let T be a tableau. If a prefixed formula $\tau{:}\psi$ of T has been introduced by applying one of the prefix generating rules except* (repl) *to a premise $\sigma{:}\varphi$ of T then we say that $\tau{:}\psi$ is generated by $\sigma{:}\varphi$, and we write $\sigma : \varphi \prec \tau{:}\psi$.*

Now we define a measure for the complexity of a prefixed formula:

Definition 9. *Let T be a tableau, $\sigma{:}\varphi$ be a prefixed formula occurring on T and let $|\varphi|$ denote the length of the φ. We define*

$$m(\sigma{:}\varphi) = 2|\varphi| + |\mathit{Set}(\sigma)|,$$

Lemma 4 (Decreasing length). *Let T be a tableau with no application of the* (repl) *rule. If $\sigma{:}\psi \prec \tau{:}\varphi$ then $m(\sigma{:}\psi) > m(\tau{:}\varphi)$.*

Proof. Assume $\sigma{:}\psi \prec \tau{:}\varphi$. We need to prove $m(\sigma{:}\psi) > 2|\varphi| + |\mathit{Set}(\tau)|$. $\tau{:}\varphi$ must have been introduced by an application of either $(\langle r\rangle)$, $([r])$, $(\textcircled{k})$, $(\neg\textcircled{k})$, $(\textcircled{e})$, $(\textcircled{r})$, $(\wedge)$ or $(\vee)$.

In the case of $(\langle r\rangle)$, $\tau{:}\varphi$ must be introduced by applying the $(\langle r\rangle)$ rule to a premise of the form $\sigma{:}\langle r\rangle\varphi$. In the case of $([r])$, $\tau{:}\varphi$ must be introduced by applying the $([r])$ rule to a pair of premises of the form $\sigma{:}[r]\varphi$, $R_r(\mathit{Label}(\sigma), \mathit{Label}(\tau))$.

In both cases we see that $\tau{:}\varphi$ is introduced by applying a rule to a formula $\sigma{:}\psi$ where $|\psi| > |\varphi|$ and where $|Set(\tau)| = |Set(\sigma)|$. Thus we get $m(\sigma{:}\psi) = |Set(\sigma)| + 2|\psi| > |Set(\tau)| + 2|\varphi|$.

If $\tau{:}\varphi$ is introduced by (ⓚ) or (¬ⓚ) from $\sigma{:}\psi$, it is immediate that $\varphi = \psi$, $|Set(\tau)| = 0$ and also that $|Set(\sigma)| > 0$, because otherwise the application would generate a prefixed formula already in the branch. Thus, $m(\sigma{:}\psi) = |Set(\sigma)| + 2|\psi| > 0 + 2|\psi| = |Set(\tau)| + 2|\varphi|$. The case of (ⓔ) is clear as the length of the set does not increase, and the length of the formula decreases. In the cases of $(\vee)$ and $(\wedge)$ the length of the formula is decreased while the set is preserved.

Finally, if $\tau{:}\varphi$ is introduced by the (ⓡ) rule from $\sigma{:}\psi$, we see that while the set may be increased by one, the length of the formula is always decremented. Then we have $m(\sigma{:}\psi) = |Set(\sigma)| + 2|\psi| > (|Set(\sigma)| + 1) + 2(|\psi| - 1) = |Set(\tau)| + 2|\varphi|$.

Lemma 5 (Finite branching). *Let Γ be a branch of a tableau. For any $\sigma{:}\varphi \in \Gamma$ there is only a finite number of prefixed formulas $\tau{:}\psi \in \Gamma$ such that $\sigma{:}\varphi \prec \tau{:}\psi$.*

Proof. For any given prefix σ the set $T_\Gamma(\sigma)$ is finite (Lemma 3), and for each formula $\varphi \in T_\Gamma(\sigma)$ at most one new prefix has been generated from σ (by applying a prefix generating rule to $\sigma{:}\varphi$). Thus $\prec$ is finitely branching.

Theorem 3. *Fix a natural number $n \geq 0$. Any tableau for $\mathcal{ML}$(ⓚ, ⓡ, ⓔ) in which the rule* (repl) *is applied at most n times per branch is finite.*

Proof. We show that any branch Γ of the tableau is finite.

Notice first that $\sigma_0{:}\varphi_0$ has no $\prec$-predecessors, and that at most $k \leq n$ other prefixed formulas of the tableau share the property of not having $\prec$-predecessors. Intuitively, each of these k formulas were introduced in Γ by the (repl) rule and hence cannot have been derived by $\prec$. We shall refer to these $k + 1$ formulas as 'generating formulas'.

It is easy to see that each generating formula induces a connected component in the graph of $\prec$. Then, every $\sigma{:}\varphi \in \Gamma$ belongs to (at least) one of these $k + 1$ connected components. As the function m decreases monotonously along any path of each of the connected components (Lemma 4), all paths of the component are finite.

By construction, there is a path between a generating formula and every node of its connected component. Then the graph is weakly connected and every path is finite. By König's Lemma the connected component is either finite or has infinite branching. As we know by Lemma 5 that it has finite branching, Γ must be finite.

Since we limit the number of applications of (repl) to n, we may have a saturated open tableau for φ whose induced model $\mathcal{M}^\Gamma$ is not a model for φ (recall that that we are taking into account the constraint on the number of applications of (repl) when talking about *saturation*). This implies that it is no longer safe to answer SAT in these cases. But we can try to identify, given a formula φ, whether $\mathcal{M}^\Gamma$ is indeed a model for φ. The algorithm we propose is outlined as follows:

1. Given a formula φ and a parameter $n \geq 0$, build T, a saturated tableau for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ with root $\langle w, \emptyset \rangle^{\mathcal{C}_{\text{all}}}{:}\varphi$ using at most n applications of the (repl) rule per branch.
2. If T is closed answer UNSAT.
3. Else, if T has an open branch Γ, compute the induced model $\mathcal{M}^{\Gamma}$.
4. If $\mathcal{M}^{\Gamma}, [w] \models \varphi$ then answer SAT.
5. Else, answer NOT-KNOWN.

Correctness of this algorithm is straightforward. Moreover, as we will show in Section 4, $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ is a fragment of $\mathcal{HL}(\downarrow)$, and therefore we can use a model checking algorithm for $\mathcal{HL}(\downarrow)$ to perform the step 4 in polynomial space.

Note that in the case the algorithm returns NOT-KNOWN, we can try refining the result running the algorithm again with a bigger n reusing the previously computed tableau, as the resulting tableau will be an *extension* to the previous one. This method allows us to approximate increasingly to a solution to the satisfiability problem of $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ in a controlled way.

4 Model Checking

In this section we will show that the complexity of the model checking problem for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ is PSpace-complete (actually the result already holds for $\mathcal{ML}(\textcircled{k}, \textcircled{r})$). To prove the lower bound we reduce the PSpace-complete satisfiability problem for Quantified Boolean Formulas (QBF) [15] to the model checking problem of $\mathcal{ML}(\textcircled{k}, \textcircled{r})$. To prove the upper bound, we show an equivalent preserving translation from formulas of $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ to formulas of the hybrid logic $\mathcal{HL}(\downarrow)$ [7,16].

This high complexity contrasts with the linear complexity (in both formula and model size) of model checking for the basic modal logic [17], and can be seen as a strengthening of the result of PSpace-hardness of $\mathcal{HL}(\downarrow)$ shown in [16] (in the sense that $\mathcal{ML}(\textcircled{k}, \textcircled{r})$ is a logic with strictly weaker expressive power than $\mathcal{HL}(\downarrow)$, but whose model checking problem is already PSpace-hard).

We start by giving a lower bound for $\mathcal{ML}(\textcircled{k}, \textcircled{r})$. Since $\mathcal{ML}(\textcircled{k}, \textcircled{r})$ is a sublanguage of $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$, the result also holds for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$.

Theorem 4. *Model checking for $\mathcal{ML}(\textcircled{k}, \textcircled{r})$ is PSpace-hard.*

Proof. We prove it by giving a polynomial-time reduction of QBF-SAT, known to be PSpace-complete [15], to the model checking problem of $\mathcal{ML}(\textcircled{k}, \textcircled{r})$.

Let α be a QBF formula with propositional variables $\{x_1, \ldots x_k\}$. Without loss of generality, we assume that α has no free-variables and no variable is quantified twice. One can build in polynomial time the relational structure $\mathcal{M}^k = \langle W, \{R_r\}, V, S \rangle$, over a signature with one relation symbol and propositions $\{p_\top, p_{x_1}, \ldots p_{x_k}\}$, where

$$\begin{array}{ll}
V(p_{x_i}) = \{w_{x_i}\} \text{ for all } i \in [1..k] & S \;= \emptyset \\
V(p_\top) \;= \{w^\top_{x_1}, w^\top_{x_2}, \ldots w^\top_{x_k}\} & W = \{w\} \cup \{w_{x_i}, w^\top_{x_i}, w^\bot_{x_i} \mid i \in [1..k]\} \\
R_r \quad\;\; = \{(w, w_{x_i}), (w_{x_i}, w^\top_{x_i}), (w^\top_{x_i}, w), (w_{x_i}, w^\bot_{x_i}), (w^\bot_{x_i}, w) \mid i \in [1..k]\}. &
\end{array}$$

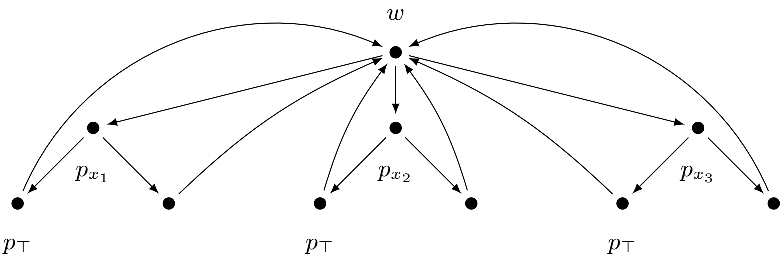


Fig. 2. $\mathcal{M}^k$ for $k = 3$

Figure 2 depicts $\mathcal{M}^k$ for $k = 3$. Let Tr be the following linear-time translation:

$$\mathsf{Tr}(x_i) := \langle r\rangle(p_{x_i} \wedge \langle r\rangle(p_\top \wedge \textcircled{k})) \qquad \mathsf{Tr}(\exists x_i.\alpha) := \langle r\rangle(p_{x_i} \wedge \langle r\rangle\textcircled{r}\langle r\rangle\mathsf{Tr}(\alpha))$$
$$\mathsf{Tr}(\neg\alpha) := \overline{\mathsf{Tr}(\alpha)} \qquad \mathsf{Tr}(\alpha \wedge \beta) := \mathsf{Tr}(\alpha) \wedge \mathsf{Tr}(\beta)$$

It only remains to see that α is satisfiable iff $\mathcal{M}^k, w \models \mathsf{Tr}(\alpha)$ holds, but this is relatively straightforward. Let us write $v \models_{\text{qbf}} \alpha$ if valuation $v : \{x_1, \ldots x_k\} \to 2$ satisfies α. For a *memory* $S \subseteq W$, define $v_S : \{x_1, \ldots x_k\}$ as "$v_S(x_i) = 1$ iff $w^\top_{x_i} \in S$". Let β be any subformula of α; we will now show by induction on β that $\langle W, V, R_r, S\rangle, w \models \mathsf{Tr}(\beta)$ iff $v_S \models_{\text{qbf}} \beta$ whenever S satisfies i) if x_i is free in β, then $w^\top_{x_i} \in S$ or $w^\perp_{x_i} \in S$ but not both, and ii) if x_i is not free in β then $w^\top_{x_i} \notin S$ and $w^\perp_{x_i} \notin S$. Observe that from here it will follow that $\mathcal{M}^k, w \models \mathsf{Tr}(\alpha)$ iff $v \models_{\text{qbf}} \alpha$ for any v (since α has no free variables) iff α is satisfiable.

For the base case, $v_S \models_{\text{qbf}} x_i$ iff $w^\top_{x_i} \in S$ which implies (from the definition of $\mathcal{M}^k$) $\langle W, V, R_r, S\rangle, w \models \mathsf{Tr}(x_i)$. For the other direction, suppose now that $\langle W, V, R_r, S\rangle, w \not\models \mathsf{Tr}(x_i)$. This means that $\langle W, V, R_r, S\rangle, w \models [r](\neg p_{x_i} \vee [r](\neg p_\top \vee \neg\textcircled{k}))$ which implies $\langle W, V, R_r, S\rangle, w_{x_i} \models [r](\neg p_\top \vee \neg\textcircled{k})$ which implies $\langle W, V, R_r, S\rangle, w^\top_{x_i} \models \neg\textcircled{k}$ and, thus, $w^\top_{x_i} \notin S$. Therefore we have $v_S \not\models_{\text{qbf}} x_i$.

Consider now the case $\beta = \exists x_i.\gamma$. Since α has no rebound variables we know $w^\top_{x_i} \notin S$ and $w^\perp_{x_i} \notin S$. We have $v_S \models_{\text{qbf}} \beta$ iff $v_S[x_i \mapsto 0] \models_{\text{qbf}} \gamma$ or $v_S[x_i \mapsto 1] \models_{\text{qbf}} \gamma$ iff $v_{S \cup \{w^\perp_{x_i}\}} \models_{\text{qbf}} \gamma$ or $v_{S \cup \{w^\top_{x_i}\}} \models_{\text{qbf}} \gamma$ iff, by inductive hypothesis, $\langle W, V, R_r, S \cup \{w^\perp_{x_i}\}\rangle \models \mathsf{Tr}(\gamma)$ or $\langle W, V, R_r, S \cup \{w^\top_{x_i}\}\rangle \models \mathsf{Tr}(\gamma)$ iff $\langle W, V, R_r, S\rangle, w^\perp_{x_i} \models \textcircled{r}\langle r\rangle\mathsf{Tr}(\gamma)$ or $\langle W, V, R_r, S\rangle, w^\top_{x_i} \models \textcircled{r}\langle r\rangle\mathsf{Tr}(\gamma)$ iff $\langle W, V, R_r, S\rangle, w \models \langle r\rangle(p_{x_i} \wedge \langle r\rangle\textcircled{r}\langle r\rangle\mathsf{Tr}(\gamma))$ iff $\langle W, V, R_r, S\rangle, w \models \mathsf{Tr}(\exists x_i.\gamma)$.

The boolean cases follow directly from the inductive hypothesis.

To see that $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ is in PSpace it is enough to show that any $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ formula can be translated to an equivalent formula of $\mathcal{H}(\downarrow)$, whose model checking problem is known to be PSpace-complete [16]. Recall that the language of $\mathcal{HL}(\downarrow)$ is the language of the basic modal logic extended with nominals and the $\downarrow$ binder (see [7] for details). Formulas of $\mathcal{HL}(\downarrow)$ are also interpreted over relational structures, but we need additionally an assignment function to interpret nominals and $\downarrow$. More formally, to evaluate a formula of $\mathcal{HL}(\downarrow)$, we need a relational structure $\mathcal{M} = \langle W, (R_r)_{r \in \mathsf{Rel}}, V\rangle$ (where W is a non empty set, each R_r is a binary relation over W, and V is a valuation function), and an assignment function g such that for any nominal i, $g(i) \in W$. Given a relational structure $\mathcal{M} = \langle W, (R_r)_{r \in \mathsf{Rel}}, V\rangle$ and an assignment g, the semantic conditions for the $\downarrow$ operator and the nominals is defined as

$$\begin{array}{ll} \mathcal{M}, g, w \models i & \text{iff } g(i) = w \\ \mathcal{M}, g, w \models {\downarrow} i.\varphi & \text{iff } \mathcal{M}, g', w \models \varphi \text{ where } g' \text{ is identical to } g \\ & \qquad\qquad \text{except perhaps in that } g'(i) = w. \end{array}$$

The semantics for the other operators is the same as for the basic modal logic. Formulas in which any nominal i appears in the scope of a binder $\downarrow i$ are called *sentences.*

In order to define a translation between $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ and $\mathcal{HL}(\downarrow)$ we have to find a mapping between the models of each logic. Since $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$-models may have a nonempty memory, we must introduce a shift in the signature of $\mathcal{HL}(\downarrow)$-models to encode this information. We will associate every $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$-model $\mathcal{M} = \langle W, (R_r)_{r \in \mathsf{Rel}}, V, S\rangle$ over the signature $\langle \mathsf{Prop}, \mathsf{Rel}\rangle$ with the $\mathcal{HL}(\downarrow)$-model $\mathcal{M}' = \langle W, (R_r)_{r \in \mathsf{Rel}}, V'\rangle$ over the signature $\langle \mathsf{Prop} \cup \{\mathit{known}\}, \mathsf{Rel}, \mathsf{Nom}\rangle$, where V' is identical to V over Prop, and $V'(\mathit{known}) = S$.

Theorem 5. *Model checking for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ is PSpace-complete.*

Proof. We define the translation Tr, taking formulas of $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ over the signature $\langle \mathsf{Prop}, \mathsf{Rel}\rangle$ to $\mathcal{HL}(\downarrow)$ sentences over the signature $\langle \mathsf{Prop} \cup \{\mathit{known}\}, \mathsf{Rel}, \mathsf{Nom}\rangle$. Tr is defined for any finite set $N \subseteq \mathsf{Nom}$ and $C \in \{\mathcal{C}_{\mathrm{all}}, \mathcal{C}_{\emptyset}\}$ as follows:

$$\begin{aligned}
\mathsf{Tr}_{N,C}(p) &= p \quad p \in \mathsf{Prop} \\
\mathsf{Tr}_{N,C}(\neg p) &= \neg p \quad p \in \mathsf{Prop} \\
\mathsf{Tr}_{N,C}(\textcircled{k}) &= \begin{cases} (\bigvee_{i \in N} i) \vee \mathit{known} & \text{if } C = \mathcal{C}_{\mathrm{all}} \\ \bigvee_{i \in N} i & \text{if } C = \mathcal{C}_{\emptyset} \end{cases} \\
\mathsf{Tr}_{N,C}(\neg\textcircled{k}) &= \begin{cases} (\bigwedge_{i \in N} \neg i) \wedge \neg\mathit{known} & \text{if } C = \mathcal{C}_{\mathrm{all}} \\ \bigwedge_{i \in N} \neg i & \text{if } C = \mathcal{C}_{\emptyset} \end{cases} \\
\mathsf{Tr}_{N,C}(\varphi_1 \wedge \varphi_2) &= \mathsf{Tr}_{N,C}(\varphi_1) \wedge \mathsf{Tr}_{N,C}(\varphi_2) \\
\mathsf{Tr}_{N,C}(\varphi_1 \vee \varphi_2) &= \mathsf{Tr}_{N,C}(\varphi_1) \vee \mathsf{Tr}_{N,C}(\varphi_2) \\
\mathsf{Tr}_{N,C}(\langle r\rangle\varphi) &= \langle r\rangle\mathsf{Tr}_{N,C}(\varphi) \\
\mathsf{Tr}_{N,C}([r]\varphi) &= [r]\mathsf{Tr}_{N,C}(\varphi) \\
\mathsf{Tr}_{N,C}(\textcircled{r}\varphi) &= {\downarrow} i.\mathsf{Tr}_{N \cup \{i\},C}(\varphi) \quad \text{where } i \notin N. \\
\mathsf{Tr}_{N,C}(\textcircled{e}\varphi) &= \mathsf{Tr}_{\emptyset,\mathcal{C}_{\emptyset}}(\varphi).
\end{aligned}$$

A simple induction shows that, given a formula $\varphi \in \mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$, $\mathcal{M}, w \models \varphi$ iff $\mathcal{M}', g, w \models \mathsf{Tr}_{\emptyset,\mathcal{C}_{\mathrm{all}}}(\varphi)$ for any g.

5 Conclusions, Related and Further Work

The family of memory logics has been introduced to investigate, in the simplest possible set up, the idea of models with a dynamic state. From that perspective they are closely related to Dynamic Epistemic Logics (DELs) as those discussed in [8] and many others [9,10,11,12]. Compared to these domain-specific logics, the goals of memory logics are humbler, focusing on developing a suitable proof and model theory for logics whose semantics is defined using models that can evolve during the evaluation of a formula. From a purely formal point of view they are closer to hybrid logics. And the logic $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ that we investigated in this paper is closely related, but expressively weaker, than the logic $\mathcal{HL}(\downarrow)$ [7].

It was already proved in [2,3] that the satisfiability problem of $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ was undecidable. In this paper we develop sound and complete tableau calculi for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ and $\mathcal{ML}(\textcircled{k}, \textcircled{r})$ (Theorems 1 and 2) which, given the undecidability result, are non terminating. By restricting the application of one of the rules in the calculi we can obtain termination at the expense of completeness (Theorem 3). To ensure soundness of this calculus we need to perform model checking whenever we obtain an open branch. Theorem 4 shows that the model checking problem for $\mathcal{ML}(\textcircled{k}, \textcircled{r})$ is PSpace-complete.

To close the paper, we discuss how the tableau calculus for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e})$ could be extended to cover another interesting memory operator. Define the *forget* operator $\textcircled{f}$ as follows:

$$\mathcal{M}, w \models \textcircled{f}\varphi \iff \langle W, (R_r)_{r \in \mathsf{Rel}}, V, S - \{w\} \rangle, w \models \varphi.$$

The $\textcircled{f}$ operator gives us a fine control on which elements we want to eliminate from the memory of the model. Prefixes in the calculus for $\mathcal{ML}(\textcircled{k}, \textcircled{r}, \textcircled{e}, \textcircled{f})$ will have to explicitly record forgotten worlds in a separate set (it is not enough to simply eliminate them from the set of remembered labels). For example, the rules for $(\textcircled{f})$ and $(\textcircled{r})$ would be

$$(\textcircled{f})\ \frac{\langle w, R, F\rangle^C{:}\textcircled{f}\varphi}{\langle w, R - \{w\}, F \cup \{w\}\rangle^C{:}\varphi} \qquad (\textcircled{r})\ \frac{\langle w, R, F\rangle^C{:}\textcircled{r}\varphi}{\langle w, R \cup \{w\}, F - \{w\}\rangle^C{:}\varphi}$$

where R is the set of remembered states and F the set of explicitly forgotten states. On the other hand, the rules for $(\textcircled{k})$ and $(\neg\textcircled{k})$ would be

$$(\textcircled{k})\ \frac{\langle w, \{v_1, \ldots v_k\}, F\rangle^C{:}\textcircled{k}}{w \approx v_1 \mid \cdots \mid w \approx v_k \mid \langle w, \emptyset, F\rangle^C{:}\textcircled{k}} \quad (\neg\textcircled{k})\ \frac{\langle w, R, \{v_1, \ldots v_k\}\rangle^C{:}\neg\textcircled{k}}{w \approx v_1 \mid \cdots \mid w \approx v_k \mid \langle w, R, \emptyset\rangle^C{:}\neg\textcircled{k}}$$

Notice the symmetry between the rules, which corresponds to the symmetry in the semantic definition of $\textcircled{r}$ and $\textcircled{f}$. Besides these changes, the tableau rules and the completeness argument remain roughly the same.

References

1. Areces, C.: Hybrid logics: The old and the new. In: Proc. of LogKCA 2007. San Sebastian, Spain (2007)
2. Areces, C., Figueira, D., Figueira, S., Mera, S.: Expressive power and decidability for memory logics. In: Hodges, W., de Queiroz, R. (eds.) Logic, Language, Information and Computation. LNCS, vol. 5110, pp. 56–68. Springer, Heidelberg (2008)
3. Areces, C., Figueira, D., Figueira, S., Mera, S.: Expressive power and decidability for memory logics. Technical report, INRIA Nancy, Grand Est. (2008); Extended version of [2]
4. Areces, C., Figueira, S., Mera, S.: Completeness results for memory logics. In: LFCS 2009. LNCS, vol. 5407. Springer, Heidelberg (2009)
5. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press, Cambridge (2001)
6. Blackburn, P., Wolter, F., van Benthem, J. (eds.): Handbook of Modal Logics. Elsevier, Amsterdam (2006)
7. Areces, C., ten Cate, B.: Hybrid logics. In: [6], pp. 821–868
8. van Ditmarsch, H., van der Hoek, W., Kooi, B.: Dynamic Epistemic Logic. Kluwer Academic Publishers, Dordrecht (2007)
9. Harel, E., Lichtenstein, O., Pnueli, A.: Explicit clock temporal logic. In: Proc. of LICS 1990, pp. 402–413 (1990)
10. Plaza, J.: Logics of public communications. In: Proc. of 4th International Symp. on Methodologies for Intelligent Systems, pp. 201–216 (1989)
11. van Benthem, J.: Logics for information update. In: Proc. of TARK 2001, pp. 51–67. Morgan Kaufmann Pub., San Francisco (2001)
12. van Benthem, J., van Eijck, J., Kooi, B.: Logics of communication and change. Information and Computation 204(11), 1620–1662 (2006)
13. Chandra, A., Merlin, P.: Optimal implementation of conjunctive queries in relational databases. In: Proc. of 9th ACM Symp. on Theory of Computing, pp. 77–90 (1977)
14. Bolander, T., Blackburn, P.: Termination for hybrid tableaus. Journal of Logic and Computation 17, 517–554 (2007)
15. Papadimitriou, C.: Computational Complexity. Addison-Wesley, Reading (1994)
16. Franceschet, M., de Rijke, M.: Model checking hybrid logics (with an application to semistructured data). Journal of Applied Logic 4(3), 279–304 (2006)
17. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)

Canonical Constructive Systems*

Arnon Avron and Ori Lahav

School of Computer Science, Tel-Aviv University, Israel
{aa,orilahav}@post.tau.ac.il

Abstract. We define the notions of a canonical inference rule and a canonical constructive system in the framework of strict single-conclusion Gentzen-type systems (or, equivalently, natural deduction systems), and develop a corresponding general non-deterministic Kripke-style semantics. We show that every constructive canonical system induces a class of non-deterministic Kripke-style frames, for which it is strongly sound and complete. This non-deterministic semantics is used to show that such a system always admits a strong form of the cut-elimination theorem, and for providing a decision procedure for such systems.

1 Introduction

The standard intuitionistic connectives ($\supset, \wedge, \vee$, and $\bot$) are of great importance in theoretical computer science, especially in type theory, where they correspond to basic operations on types (via the formulas-as-types principle and Curry-Howard isomorphism). Now a natural question is: what is so special about these connectives? The standard answer is that they are all *constructive* connectives. But then what exactly is a constructive connective, and can we define other basic constructive connectives beyond the four intuitionistic ones? And what does the last question mean anyway: how do we "define" new (or old) connectives?

Concerning the last question there is a long tradition starting from [10] (see e.g. [14] for discussions and references) according to which the meaning of a connective is determined by the introduction and elimination rules which are associated with it. Here one usually has in mind natural deduction systems of an ideal type, where each connective has its own introduction and elimination rules, and these rules should meet the following conditions: in a rule for some connective this connective should be mentioned exactly once, and no other connective should be involved. The rule should also be pure in the sense of [1] (i.e., there should be no side conditions limiting its application), and its active formulas should be immediate subformulas of its principal formula.

Unfortunately, already the handling of negation requires rules which are not ideal in this sense. For intuitionistic logic this problem has been solved by not taking negation as a basic constructive connective, but defining it instead in terms of more basic connectives that can be characterized by "ideal" rules ($\neg\varphi$

* This research was supported by THE ISRAEL SCIENCE FOUNDATION (grant No 809-06).

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 62–76, 2009.

is defined as $\varphi \rightarrow\perp$). For classical logic the problem was solved by Gentzen himself by moving to what is now known as Gentzen-type systems or sequential calculi. These calculi employ single-conclusion sequents in their intuitionistic version, and multiple-conclusion sequents in their classical version. Instead of introduction and elimination rules they use left introduction rules and right introduction rules. The intuitive notions of an "ideal rule" can be adapted to such systems in a straightforward way, and it is well known that the usual classical connectives and the basic intuitionistic connectives can indeed be fully characterized by "ideal" Gentzen-type rules. Moreover: although this can be done in several ways, in all of them the cut-elimination theorem obtains.

For the multiple-conclusion framework these facts were considerably generalized in [5,6] by defining "multiple-conclusion canonical propositional Gentzen-type rules and systems" in precise terms. A constructive necessary and sufficient *coherence* criterion for the non-triviality of such systems was then provided, and it was shown that a system of this kind admits cut-elimination iff it is coherent. It was further proved that the semantics of such systems is provided by two-valued non-deterministic matrices (two-valued Nmatrices) — a natural generalization of the classical truth-tables. In fact, a characteristic two-valued Nmatrix was constructed for every coherent canonical propositional system. That work shows that there is a large family of what may be called semi-classical connectives (which includes all the classical connectives), each of which has both a proof-theoretical characterization in terms of a coherent set of canonical (= "ideal") rules, and a semantic characterization using two-valued Nmatrices.

In this paper we develop a similar theory for the constructive propositional framework. We define the notions of a canonical rule and a canonical system in the framework of strict single-conclusion Gentzen-type systems (or, equivalently, natural deduction systems). We prove that here too a canonical system is non-trivial iff it is coherent (where coherence is a constructive condition, defined like in the multiple-conclusion case). We develop a general non-deterministic Kripke-style semantics for such systems, and show that every constructive canonical system (i.e. coherent canonical single-conclusion system) induces a class of non-deterministic Kripke-style frames for which it is strongly sound and complete. We use this non-deterministic semantics to show that all constructive canonical systems admit a strong form of the cut-elimination theorem. We also use it for providing decision procedures for all such systems. These results again identify a large family of basic constructive connectives, each having both a proof-theoretical characterization in terms of a coherent set of canonical rules, and a semantic characterization using non-deterministic frames. The family includes the standard intuitionistic connectives ($\supset, \wedge, \vee$, and $\perp$), as well as many other independent connectives.

2 Canonical Constructive Systems

In what follows $\mathcal{L}$ is a propositional language, $\mathcal{F}$ is its set of wffs, p, q, r denote atomic formulas, ψ, φ, θ denote arbitrary formulas (of $\mathcal{L}$), T, S denote subsets of $\mathcal{F}$, and $\Gamma, \Delta, \Sigma, \Pi$ denote finite subsets of $\mathcal{F}$. We assume that the atomic formulas of $\mathcal{L}$ are $p_1, p_2, \ldots$ (in particular: $\{p_1, p_2, \ldots, p_n\}$ are the first n atomic formulas of $\mathcal{L}$).

Definition 1. A *Tarskian consequence relation* (*tcr* for short) for $\mathcal{L}$ is a binary relation $\vdash$ between sets of formulas of $\mathcal{L}$ and formulas of $\mathcal{L}$ that satisfies the following conditions:

strong reflexivity: if $\varphi \in T$ then $T \vdash \varphi$.
monotonicity: if $T \vdash \varphi$ and $T \subseteq T'$ then $T' \vdash \varphi$.
transitivity (*cut*): if $T \vdash \psi$ and $T, \psi \vdash \varphi$ then $T \vdash \varphi$.

Definition 2. A *substitution* in $\mathcal{L}$ is a function σ from the atomic formulas to the set of formulas of $\mathcal{L}$. σ is extended to formulas and sets of formulas in the obvious way.

Definition 3. A tcr $\vdash$ for $\mathcal{L}$ is *structural* if for every substitution σ and every T and φ, if $T \vdash \varphi$ then $\sigma(T) \vdash \sigma(\varphi)$. $\vdash$ is *finitary* if the following condition holds for all T and φ: if $T \vdash \varphi$ then there exists a finite $\Gamma \subseteq T$ such that $\Gamma \vdash \varphi$. $\vdash$ is *consistent* (or *non-trivial*) if $p_1 \not\vdash p_2$.

It is easy to see (see [6]) that there are exactly two inconsistent structural tcrs in any given language[1]. These tcrs are obviously trivial, so we exclude them from our definition of a *logic*:

Definition 4. A propositional *logic* is a pair $\langle \mathcal{L}, \vdash \rangle$, where $\mathcal{L}$ is a propositional language, and $\vdash$ is a tcr for $\mathcal{L}$ which is structural, finitary, and consistent.

Since a finitary consequence relation $\vdash$ is determined by the set of pairs $\langle \Gamma, \varphi \rangle$ such that $\Gamma \vdash \varphi$, it is natural to base proof systems for logics on the use of such pairs. This is exactly what is done in natural deduction systems and in (strict) single-conclusion Gentzen-type systems (both introduced in [10]). Formally, such systems manipulate objects of the following type:

Definition 5. A *sequent* is an expression of the form $\Gamma \Rightarrow \Delta$ where Γ and Δ are finite sets of formulas, and Δ is either a singleton or empty. A sequent of the form $\Gamma \Rightarrow \{\varphi\}$ is called *definite*, and we shall denote it by $\Gamma \Rightarrow \varphi$. A sequent of the form $\Gamma \Rightarrow \{\}$ is called *negative*, and we shall denote it by $\Gamma \Rightarrow$. A Horn *clause* is a sequent which consists of atomic formulas only.

Note. Natural deduction systems, and the strict single-conclusion Gentzen-type systems investigated in this paper, manipulate only definite sequents in their derivations. However, negative sequents may be used in the formulations of their rules (in the form of negative Horn clauses).

The following definitions formulate in exact terms the idea of an "ideal rule" which was described in the introduction:

[1] In one $T \vdash \varphi$ for every T and φ, in the other $T \vdash \varphi$ for every nonempty T and φ.

Definition 6

1. A *canonical introduction rule* is an expression of the form:

$$\{\Pi_i \Rightarrow \Sigma_i\}_{1\leq i\leq m} / \ \Rightarrow \diamond(p_1, p_2, \ldots, p_n)$$

where $m \geq 0$, $\diamond$ is a connective of arity n, and for all $1 \leq i \leq m$, $\Pi_i \Rightarrow \Sigma_i$ is a *definite* Horn clause such that $\Pi_i \cup \Sigma_i \subseteq \{p_1, p_2, \ldots, p_n\}$.
2. A *canonical elimination rule*[2] is an expression of the form

$$\{\Pi_i \Rightarrow \Sigma_i\}_{1\leq i\leq m} / \ \diamond(p_1, p_2, \ldots, p_n) \Rightarrow$$

where $m \geq 0$, $\diamond$ is a connective of arity n, and for all $1 \leq i \leq m$, $\Pi_i \Rightarrow \Sigma_i$ is a Horn clause (either definite or negative) such that $\Pi_i \cup \Sigma_i \subseteq \{p_1, p_2, \ldots, p_n\}$.
3. An *application* of the rule $\{\Pi_i \Rightarrow \Sigma_i\}_{1\leq i\leq m} / \ \Rightarrow \diamond(p_1, p_2, \ldots, p_n)$ is any inference step of the form:

$$\frac{\{\Gamma, \sigma(\Pi_i) \Rightarrow \sigma(\Sigma_i)\}_{1\leq i\leq m}}{\Gamma \Rightarrow \diamond(\sigma(p_1), \ldots, \sigma(p_n))}$$

where Γ is a finite set of formulas and σ is a substitution in $\mathcal{L}$.
4. An *application* of the rule $\{\Pi_i \Rightarrow \Sigma_i\}_{1\leq i\leq m} / \ \diamond(p_1, p_2, \ldots, p_n) \Rightarrow$ is any inference step of the form:

$$\frac{\{\Gamma, \sigma(\Pi_i) \Rightarrow \sigma(\Sigma_i), E_i\}_{1\leq i\leq m}}{\Gamma, \diamond(\sigma(p_1), \ldots, \sigma(p_n)) \Rightarrow \theta}$$

where Γ and σ are as above, θ is a formula, and for all $1 \leq i \leq m$: $E_i = \theta$ in case Σ_i is empty, and E_i is empty otherwise.

Note. We formulated the definition above in terms of Gentzen-type systems. However, we could have formulated them instead in terms of natural deduction systems. The definition of an application of an introduction rule is defined in this context exactly as above, while an application of an elimination rule of the form $\{\Pi_i \Rightarrow \Sigma_i\}_{1\leq i\leq m} / \ \diamond(p_1, p_2, \ldots, p_n) \Rightarrow$ is in the context of natural deduction any inference step of the form:

$$\frac{\{\Gamma, \sigma(\Pi_i) \Rightarrow \sigma(\Sigma_i), E_i\}_{1\leq i\leq m} \qquad \Gamma \Rightarrow \diamond(\sigma(p_1), \ldots, \sigma(p_n))}{\Gamma \Rightarrow \theta}$$

where Γ, σ, θ and E_i are as above.

Here are some examples of well-known canonical rules:

[2] The introduction/elimination terminology is due to the natural deduction context. For the Gentzen-type context the names "right introduction rule" and "left introduction rule" might be more appropriate, but we prefer to use a uniform terminology.

Conjunction. The two usual rules for conjunction are:

$$\{p_1, p_2 \Rightarrow \ \} \ / \ p_1 \wedge p_2 \Rightarrow \quad \text{and} \quad \{ \Rightarrow p_1 \ , \ \Rightarrow p_2\} \ / \ \Rightarrow p_1 \wedge p_2$$

In the Gentzen-type context applications of these rules have the form:

$$\frac{\Gamma, \psi, \varphi \Rightarrow \theta}{\Gamma, \psi \wedge \varphi \Rightarrow \theta} \qquad \frac{\Gamma \Rightarrow \psi \quad \Gamma \Rightarrow \varphi}{\Gamma \Rightarrow \psi \wedge \varphi}$$

In natural deduction systems applications of the first have the form:

$$\frac{\Gamma, \psi, \varphi \Rightarrow \theta \quad \Gamma \Rightarrow \psi \wedge \varphi}{\Gamma \Rightarrow \theta}$$

The above elimination rule can easily be shown to be equivalent to the combination of the two more usual elimination rules for conjunction.

Implication. The two usual rules for implication are:

$$\{\Rightarrow p_1 \ , \ p_2 \Rightarrow\} \ / \ p_1 \supset p_2 \Rightarrow \quad \text{and} \quad \{p_1 \Rightarrow p_2\} \ / \ \Rightarrow p_1 \supset p_2$$

In the Gentzen-type context applications of these rules have the form:

$$\frac{\Gamma \Rightarrow \psi \quad \Gamma, \varphi \Rightarrow \theta}{\Gamma, \psi \supset \varphi \Rightarrow \theta} \qquad \frac{\Gamma, \psi \Rightarrow \varphi}{\Gamma \Rightarrow \psi \supset \varphi}$$

In natural-deduction systems applications of the first have the form:

$$\frac{\Gamma \Rightarrow \psi \quad \Gamma, \varphi \Rightarrow \theta \quad \Gamma \Rightarrow \psi \supset \varphi}{\Gamma \Rightarrow \theta}$$

Again this form of the rule is obviously equivalent to the more usual one (from $\Gamma \Rightarrow \psi$ and $\Gamma \Rightarrow \psi \supset \varphi$ infer $\Gamma \Rightarrow \varphi$).

Absurdity. In intuitionistic logic there is no introduction rule for the absurdity constant $\bot$, and there is exactly one elimination rule for it: $\{\} \ / \ \bot \Rightarrow$. In the Gentzen-type context applications of this rule provide new *axioms*: $\Gamma, \bot \Rightarrow \varphi$. In natural-deduction systems applications of the same rule allow us to infer $\Gamma \Rightarrow \varphi$ from $\Gamma \Rightarrow \bot$.

Semi-implication. Consider the "semi-implication" $\leadsto$ with the following two rules:[3]

$$\{\Rightarrow p_1 \ , \ p_2 \Rightarrow\} \ / \ p_1 \leadsto p_2 \Rightarrow \quad \text{and} \quad \{\Rightarrow p_2\} \ / \ \Rightarrow p_1 \leadsto p_2$$

In the Gentzen-type context applications of these rules have the form:

$$\frac{\Gamma \Rightarrow \psi \quad \Gamma, \varphi \Rightarrow \theta}{\Gamma, \psi \leadsto \varphi \Rightarrow \theta} \qquad \frac{\Gamma \Rightarrow \varphi}{\Gamma \Rightarrow \psi \leadsto \varphi}$$

Again in natural-deduction systems applications of the first rule are equivalent to MP for $\leadsto$ (from $\Gamma \Rightarrow \psi$ and $\Gamma \Rightarrow \psi \leadsto \varphi$ infer $\Gamma \Rightarrow \varphi$).

[3] This connective was introduced in [11] for different purposes.

From now on we shall concentrate on single-conclusion Gentzen-type systems (translating our notions and results to natural deduction systems is easy).

Definition 7. A single-conclusion Gentzen-type system is called *canonical* if its axioms are the sequents of the form $\varphi \Rightarrow \varphi$, cut (from $\Gamma \Rightarrow \varphi$ and $\Delta, \varphi \Rightarrow \psi$ infer $\Gamma, \Delta \Rightarrow \psi$) and weakening (from $\Gamma \Rightarrow \psi$ infer $\Gamma, \Delta \Rightarrow \psi$) are among its rules, and each of its other rules is either a canonical introduction rule or a canonical elimination rule.

Definition 8. Let **G** be a canonical Gentzen-type system.

1. $\mathcal{S} \vdash^{seq}_{\mathbf{G}} s$ (where s is a sequent and $\mathcal{S}$ is a set of sequents) if there is a derivation in **G** of s from $\mathcal{S}$.
2. The tcr $\vdash_{\mathbf{G}}$ between *formulas* which is induced by **G** is defined by: $T \vdash_{\mathbf{G}} \varphi$ iff there exists a finite $\Gamma \subseteq T$ such that $\vdash^{seq}_{\mathbf{G}} \Gamma \Rightarrow \varphi$.

Proposition 1. *$T \vdash_{\mathbf{G}} \varphi$ iff $\{\Rightarrow \psi \mid \psi \in T\} \vdash^{seq}_{\mathbf{G}} \Rightarrow \varphi$.*

Proposition 2. *If* **G** *is canonical then $\vdash_{\mathbf{G}}$ is a structural and finitary tcr.*

The last proposition does not guarantee that every canonical system induces a *logic* (see Definition 4). For this the system should satisfy one more condition:

Definition 9. A set $\mathcal{R}$ of canonical rules for an n-ary connective $\diamond$ is called *coherent* if $S_1 \cup S_2$ is classically inconsistent (and so the empty clause can be derived from it using cuts) whenever $\mathcal{R}$ contains both $S_1/ \diamond (p_1, p_2, \ldots, p_n) \Rightarrow$ and $S_2/ \ \Rightarrow \diamond(p_1, p_2, \ldots, p_n)$.

Examples

- All the sets of rules for the connectives $\wedge, \supset, \bot$, and $\leadsto$ which were introduced in the examples above are coherent. For example, for the two rules for conjunction we have $S_1 = \{p_1, p_2 \Rightarrow \ \}$, $S_2 = \{ \ \Rightarrow p_1 \ , \ \Rightarrow p_2\}$, and $S_1 \cup S_2$ is the classically inconsistent set $\{p_1, p_2 \Rightarrow \ , \ \Rightarrow p_1 \ , \ \Rightarrow p_2\}$ (from which the empty sequent can be derived using two cuts).
- In [13] Prior introduced a "connective" T (which he called "Tonk") with the following rules: $\{p_1 \Rightarrow \ \} \ / \ p_1 T p_2 \Rightarrow$ and $\{ \ \Rightarrow p_2\} \ / \ \Rightarrow p_1 T p_2$. Prior then used "Tonk" to infer everything from everything (trying to show by this that rules alone cannot define a connective). Now the union of the sets of premises of these two rules is $\{p_1 \Rightarrow \ , \ \Rightarrow p_2\}$, and this is a classically consistent set of clauses. It follows that Prior's set of rules for Tonk is incoherent.

Definition 10. A canonical single-conclusion Gentzen-type system, **G**, is called *coherent* if every primitive connective of the language of **G** has in **G** a coherent set of rules.

Theorem 1. *Let* **G** *be a canonical Gentzen-type system. $\langle \mathcal{L}, \vdash_{\mathbf{G}} \rangle$ is a logic (i.e. $\vdash_{\mathbf{G}}$ is structural, finitary and consistent) iff* **G** *is coherent.*

Proof. Proposition 2 ensures that $\vdash_{\mathbf{G}}$ is a structural and finitary tcr.

That the coherence of **G** implies the consistency of the *multiple* conclusion consequence relation which is naturally induced by **G** was shown in [5,6]. That consequence relation extends $\vdash_{\mathbf{G}}$, and therefore also the latter is consistent.

For the converse, assume that **G** is incoherent. This means that **G** includes two rules $S_1/ \diamond (p_1, \ldots, p_n) \Rightarrow$ and $S_2/ \Rightarrow \diamond(p_1, \ldots, p_n)$, such that the set of clauses $S_1 \cup S_2$ is classically satisfiable. Let v be an assignment in $\{t, f\}$ that satisfies all the clauses in $S_1 \cup S_2$. Define a substitution σ by:

$$\sigma(p) = \begin{cases} p_{n+1} & v(p) = f \\ p & v(p) = t \end{cases}$$

Let $\Pi \Rightarrow q \in S_1 \cup S_2$. Then $\vdash_{G}^{seq} p_1, \ldots, p_n, \sigma(\Pi) \Rightarrow \sigma(q)$. This is trivial in case $v(q) = t$, since in this case $\sigma(q) = q \in \{p_1, \ldots, p_n\}$. On the other hand, if $v(q) = f$ then $v(p) = f$ for some $p \in \Pi$ (since v satisfies the clause $\Pi \Rightarrow q$). Therefore in this case $\sigma(p) = \sigma(q) = p_{n+1}$, and so again $p_1, \ldots, p_n, \sigma(\Pi) \Rightarrow \sigma(q)$ is trivially derived from an axiom. We can similarly prove that $\vdash_{\mathbf{G}}^{seq} p_1, \ldots, p_n, \sigma(\Pi) \Rightarrow p_{n+1}$ in case $\Pi \Rightarrow \in S_1 \cup S_2$. Now by applying $S_1/ \diamond (p_1, \ldots, p_n) \Rightarrow$ and $S_2/ \Rightarrow \diamond(p_1, \ldots, p_n)$ to these provable sequents we get proofs in **G** of $p_1, \ldots, p_n \Rightarrow \diamond(\sigma(p_1), \ldots, \sigma(p_n))$ and of $p_1, \ldots, p_n, \diamond(\sigma(p_1), \ldots, \sigma(p_n)) \Rightarrow p_{n+1}$. That $\vdash_{\mathbf{G}}^{seq} p_1, \ldots, p_n \Rightarrow p_{n+1}$ then follows using a cut. This easily entails that $p_1 \vdash_{\mathbf{G}} p_2$, and hence $\vdash_{\mathbf{G}}$ is not consistent. □

Note. The last theorem implies that coherence is a minimal demand from any acceptable canonical system **G**. It follows that not every set of such rules is legitimate for defining constructive connectives - only coherent ones do (and this is what is wrong with "Tonk"). Accordingly we define:

Definition 11. A *canonical constructive system* is a coherent canonical single-conclusion Gentzen-type system.

The following definition will be needed in the sequel:

Definition 12. Let $\mathcal{S}$ be a set of sequents.

1. A cut is called an $\mathcal{S}$-cut if the cut formula occurs in $\mathcal{S}$.
2. We say that there exists in a system **G** an $\mathcal{S}$-cut-free proof of a sequent s from a set of sequents $\mathcal{S}$ iff there exists a proof of s from $\mathcal{S}$ in **G** where all cuts are $\mathcal{S}$-cuts.
3. ([2]) A system **G** admits strong cut-elimination iff whenever $\mathcal{S} \vdash_{\mathbf{G}}^{seq} s$, there exists an $\mathcal{S}$-cut-free proof of s from $\mathcal{S}$.[4]

[4] By cut-elimination we mean here just the existence of proofs without (certain forms of) cuts, rather than an algorithm to transform a given proof to a cut-free one (for the assumptions-free case the term "cut-admissibility" is sometimes used).

3 Semantics for Canonical Constructive Systems

The most useful semantics for propositional intuitionistic logic (the paradigmatic constructive logic) is that of Kripke frames. In this section we generalize this semantics to arbitrary canonical constructive systems. For this we should introduce *non-deterministic* Kripke frames.[5]

Definition 13. A generalized $\mathcal{L}$*-frame* is a triple $\mathcal{W} = \langle W, \leq, v \rangle$ such that:

1. $\langle W, \leq \rangle$ is a nonempty partially ordered set.
2. v is a function from $\mathcal{F}$ to the set of persistent functions from W into $\{t, f\}$ (A function $h : W \rightarrow \{t, f\}$ is *persistent* if $h(a) = t$ implies that $h(b) = t$ for every $b \in W$ such that $a \leq b$).

Notation: We shall usually write $v(a, \varphi)$ instead of $v(\varphi)(a)$.

Definition 14. A generalized $\mathcal{L}$-frame $\langle W, \leq, v \rangle$ is a *model* of a formula φ if $v(\varphi) = \lambda a \in W.t$ (i.e.: $v(a, \varphi) = t$ for every $a \in W$). It is a model of a theory T if it is a model of every $\varphi \in T$.

Definition 15. Let $\mathcal{W} = \langle W, \leq, v \rangle$ be a generalized $\mathcal{L}$-frame, and let $a \in W$.

1. A sequent $\Gamma \Rightarrow \varphi$ is *locally true* in a if either $v(a, \psi) = f$ for some $\psi \in \Gamma$, or $v(a, \varphi) = t$.
2. A sequent $\Gamma \Rightarrow \varphi$ is *true* in a if it is locally true in every $b \geq a$.
3. A sequent $\Gamma \Rightarrow$ is *(locally) true* in a if $v(a, \psi) = f$ for some $\psi \in \Gamma$.
4. $\mathcal{W}$ is a *model* of a sequent s (either of the form $\Gamma \Rightarrow \varphi$ or $\Gamma \Rightarrow$) if s is true in every $a \in W$ (iff s is locally true in every $a \in W$). It is a model of a set of sequents $\mathcal{S}$ if it is a model of every $s \in \mathcal{S}$.

Note. $\mathcal{W}$ is a model of a formula φ iff it is a model of the sequent $\Rightarrow \varphi$.

Definition 16. Let $\langle W, \leq, v \rangle$ be a generalized $\mathcal{L}$-frame. A substitution σ in $\mathcal{L}$ satisfies a Horn clause $\Pi \Rightarrow \Sigma$ in $a \in W$ if $\sigma(\Pi) \Rightarrow \sigma(\Sigma)$ is true in a.

Note. Because of the persistence condition, a definite Horn clause of the form $\Rightarrow q$ is satisfied in a by σ iff $v(a, \sigma(q)) = t$.

Definition 17. Let $\mathcal{W} = \langle W, \leq, v \rangle$ be a generalized $\mathcal{L}$-frame, and let $\diamond$ be an n-ary connective of $\mathcal{L}$.

1. $\mathcal{W}$ *respects* an introduction rule r for $\diamond$ if $v(a, \diamond(\psi_1, \ldots, \psi_n)) = t$ whenever all the premises of r are satisfied in a by a substitution σ such that $\sigma(p_i) = \psi_i$ for $1 \leq i \leq n$ (The values of $\sigma(q)$ for $q \notin \{p_1, \ldots, p_n\}$ are immaterial here).
2. $\mathcal{W}$ *respects* an elimination rule r for $\diamond$ if $v(a, \diamond(\psi_1, \ldots, \psi_n)) = f$ whenever all the premises of r are satisfied in a by a substitution σ such that $\sigma(p_i) = \psi_i$ $(1 \leq i \leq n)$.

[5] Another type of non-deterministic (intuitionistic) Kripke frames, based on 3-valued and 4-valued non-deterministic matrices, was used in [3,4]. Non-deterministic modal Kripke frames were recently used in [9].

3. Let $\mathbf{G}$ be a canonical Gentzen-type system for $\mathcal{L}$. $\mathcal{W}$ is $\mathbf{G}$-legal if it respects all the rules of $\mathbf{G}$.

Examples

- By definition, a generalized $\mathcal{L}$-frame $\mathcal{W} = \langle W, \leq, v\rangle$ respects the rule $(\supset\Rightarrow)$ iff for every $a \in W$, $v(a, \varphi \supset \psi) = f$ whenever $v(b, \varphi) = t$ for every $b \geq a$ and $v(a, \psi) = f$. Because of the persistence condition, this is equivalent to: $v(a, \varphi \supset \psi) = f$ whenever $v(a, \varphi) = t$ and $v(a, \psi) = f$. Again by the persistence condition, this is equivalent to: $v(a, \varphi \supset \psi) = f$ whenever there exists $b \geq a$ such that $v(b, \varphi) = t$ and $v(b, \psi) = f$. $\mathcal{W}$ respects $(\Rightarrow\supset)$ iff for every $a \in W$, $v(a, \varphi \supset \psi) = t$ whenever for every $b \geq a$, either $v(b, \varphi) = f$ or $v(b, \psi) = t$. Hence the two rules together impose exactly the well-known Kripke semantics for intuitionistic implication ([12]).
- A generalized $\mathcal{L}$-frame $\mathcal{W} = \langle W, \leq, v\rangle$ respects the rule $(\leadsto\Rightarrow)$ under the same conditions it respects $(\supset\Rightarrow)$. $\mathcal{W}$ respects $(\Rightarrow\leadsto)$ iff for every $a \in W$, $v(a, \varphi \leadsto \psi) = t$ whenever $v(a, \psi) = t$ (recall that this is equivalent to: $v(b, \psi) = t$ for every $b \geq a$). Note that in this case the two rules for $\leadsto$ do not always determine the value assigned to $\varphi \leadsto \psi$: if $v(a, \psi) = f$, and there is no $b \geq a$ such that $v(b, \varphi) = t$ and $v(b, \psi) = f$, then $v(a, \varphi \leadsto \psi)$ is free to be either t or f. So the semantics of this connective is non-deterministic.
- A generalized $\mathcal{L}$-frame $\mathcal{W} = \langle W, \leq, v\rangle$ respects the rule $(T\Rightarrow)$ (see second example after Definition 9) if $v(a, \varphi T \psi) = f$ whenever $v(a, \varphi) = f$. It respects $(\Rightarrow T)$ if $v(a, \varphi T \psi) = t$ whenever $v(a, \psi) = t$. The two constraints contradict each other in case both $v(a, \varphi) = f$ and $v(a, \psi) = t$. This is a semantic explanation why Prior's "connective" T ("Tonk") is meaningless.

Definition 18. Let $\mathbf{G}$ be a canonical constructive system.

1. $\mathcal{S} \models^{seq}_{\mathbf{G}} s$ (where $\mathcal{S}$ is a set of sequents and s is a sequent) iff every $\mathbf{G}$-legal model of $\mathcal{S}$ is also a model of s.
2. The semantic tcr $\models_{\mathbf{G}}$ between *formulas* which is induced by $\mathbf{G}$ is defined by: $T \models_{\mathbf{G}} \varphi$ if every $\mathbf{G}$-legal model of T is also a model of φ.

Again we have:

Proposition 3. *$T \models_{\mathbf{G}} \varphi$ iff $\{\Rightarrow \psi \mid \psi \in T\} \models^{seq}_{\mathbf{G}} \Rightarrow \varphi$.*

4 Soundness, Completeness, Cut-Elimination

In this section we show that the two logics induced by a canonical constructive system $\mathbf{G}$ ($\vdash_{\mathbf{G}}$ and $\models_{\mathbf{G}}$) are identical. Half of this identity is given in the following theorem:

Theorem 2. *Every canonical constructive system* $\mathbf{G}$ *is strongly sound with respect to the semantics of* $\mathbf{G}$*-legal generalized frames. In other words:*

1. *If $T \vdash_{\mathbf{G}} \varphi$ then $T \models_{\mathbf{G}} \varphi$.*
2. *If $\mathcal{S} \vdash_{\mathbf{G}}^{seq} s$ then $\mathcal{S} \models_{\mathbf{G}}^{seq} s$.*

Proof. We prove the second part first. Assume that $\mathcal{S} \vdash_{\mathbf{G}}^{seq} s$, and $\mathcal{W} = \langle W, \leq, v\rangle$ is a **G**-legal model of $\mathcal{S}$. We show that s is locally true in every $a \in W$. Since the axioms of G and the premises of $\mathcal{S}$ trivially have this property, and the cut and weakening rules obviously preserve it, it suffices to show that the property of being locally true is preserved also by applications of the logical rules of **G**.

- First we deal with the elimination rules of **G**. Suppose $\Gamma, \diamond(\psi_1, \ldots, \psi_n) \Rightarrow \theta$ is derived from $\{\Gamma, \sigma(\Pi_i) \Rightarrow \sigma(\Sigma_i)\}_{1\leq i \leq m_1}$ and $\{\Gamma, \sigma(\Pi_i) \Rightarrow \theta\}_{m_1+1\leq i\leq m}$, using the elimination rule $r = \{\Pi_i \Rightarrow \Sigma_i\}_{1\leq i\leq m}/ \ \diamond(p_1, p_2, \ldots, p_n) \Rightarrow$ (where Σ_i is empty for $m_1 + 1 \leq i \leq m$, and σ is a substitution such that $\sigma(p_j) = \psi_j$ for $1 \leq j \leq n$). Assume that all the premises of this application have the required property. Let $a \in W$. If $v(a, \psi) = f$ for some $\psi \in \Gamma$ or $v(a, \theta) = t$, then we are done. Assume otherwise. Then $v(a, \theta) = f$, and (by the persistence condition) $v(b, \psi) = t$ for every $\psi \in \Gamma$ and $b \geq a$. Hence our assumption concerning $\{\Gamma, \sigma(\Pi_i) \Rightarrow \sigma(\Sigma_i)\}_{1\leq i\leq m_1}$ entails that for every $b \geq a$ and $1 \leq i \leq m_1$, either $v(b, \psi) = f$ for some $\psi \in \sigma(\Pi_i)$, or $v(b, \sigma(\Sigma_i)) = t$. This immediately implies that every definite premise of the rule is satisfied in a by σ. Since $v(a, \theta) = f$, our assumption concerning $\{\Gamma, \sigma(\Pi_i) \Rightarrow \theta\}_{m_1+1\leq i\leq m}$ entails that for every $m_1 + 1 \leq i \leq m$, $v(a, \psi) = f$ for some $\psi \in \sigma(\Pi_i)$. Hence the negative premises of the rule are also satisfied in a by σ. Since $\mathcal{W}$ respects r, it follows that $v(a, \diamond(\psi_1, \ldots, \psi_n)) = f$, as required.
- Dealing with the introduction rules is easier, and it is left for the reader.

The first part follows from the second by Propositions 1 and 3. □

For the converse, we first prove the following key result.

Theorem 3. *Let* **G** *be a canonical constructive system in $\mathcal{L}$, and let $\mathcal{S} \cup \{s\}$ be a set of sequents in $\mathcal{L}$. Then either there is an $\mathcal{S}$-cut-free proof of s from $\mathcal{S}$, or there is a* **G***-legal model of $\mathcal{S}$ which is not a model of s.*

Proof. **(outline)** Assume that $s = \Gamma_0 \Rightarrow \varphi_0$ does not have an $\mathcal{S}$-cut-free proof in **G**. Let $\mathcal{F}'$ be the set of subformulas of $\mathcal{S} \cup \{s\}$. Given a formula $\varphi \in \mathcal{F}'$, call a theory $\mathcal{T} \subseteq \mathcal{F}'$ *φ-maximal* if there is no finite $\Gamma \subseteq \mathcal{T}$ such that $\Gamma \Rightarrow \varphi$ has an $\mathcal{S}$-cut-free-proof from $\mathcal{S}$, but every proper extension $\mathcal{T}' \subseteq \mathcal{F}'$ of $\mathcal{T}$ contains such a finite subset Γ. Obviously, if $\Gamma \subseteq \mathcal{F}'$, $\varphi \in \mathcal{F}'$ and $\Gamma \Rightarrow \varphi$ has no $\mathcal{S}$-cut-free-proof from $\mathcal{S}$, then Γ can be extended to a theory $\mathcal{T} \subseteq \mathcal{F}'$ which is φ-maximal. In particular: Γ_0 can be extended to a φ_0-maximal theory $\mathcal{T}_0$.

Now let $\mathcal{W} = \langle W, \subseteq, v\rangle$, where:

- W is the set of all extensions of $\mathcal{T}_0$ in $\mathcal{F}'$ which are φ-maximal for some $\varphi \in \mathcal{F}'$.
- v is defined inductively as follows. For atomic formulas:

$$v(\mathcal{T}, p) = \begin{cases} t & p \in \mathcal{T} \\ f & p \notin \mathcal{T} \end{cases}$$

Suppose $v(\mathcal{T}, \psi_i)$ has been defined for all $\mathcal{T} \in W$ and $1 \leq i \leq n$. We let $v(\mathcal{T}, \diamond(\psi_1, \ldots, \psi_n)) = t$ iff at least one of the following holds:

1. There exists an introduction rule for $\diamond$ whose set of premises is satisfied in $\mathcal{T}$ by a substitution σ such that $\sigma(p_i) = \psi_i$ $(1 \leq i \leq n)$.
2. $\diamond(\psi_1, \ldots, \psi_n) \in \mathcal{T}$ and there does not exist $\mathcal{T}' \in W$, $\mathcal{T} \subseteq \mathcal{T}'$, and an elimination rule for $\diamond$ whose set of premises is satisfied in $\mathcal{T}'$ by a substitution σ such that $\sigma(p_i) = \psi_i$ $(1 \leq i \leq n)$.[6]

First we prove that $\mathcal{W}$ is a generalized $\mathcal{L}$-frame:

- W is not empty because $\mathcal{T}_0 \in W$.
- That v is persistent is proved by structural induction.

Next we prove that $\mathcal{W}$ is **G**-legal:

1. The introduction rules are directly respected by the first condition in v's definition.
2. Let r be an elimination rule for $\diamond$, and suppose all its premises are satisfied in some $\mathcal{T} \in W$ by a substitution σ such that $\sigma(p_i) = \psi_i$. Then neither of the conditions under which $v(\mathcal{T}, \diamond(\psi_1, \ldots, \psi_n)) = t$ can hold: the second by definition, and the first because of **G**'s coherence.

It remains to prove that $\mathcal{W}$ is a model of $\mathcal{S}$ but not of s. For this we first prove that the following hold for every $\mathcal{T} \in W$ and every formula $\psi \in \mathcal{F}'$:

(a) If $\psi \in \mathcal{T}$ then $v(\mathcal{T}, \psi) = t$.
(b) If $\mathcal{T}$ is ψ-maximal then $v(\mathcal{T}, \psi) = f$.

(a) and **(b)** are proved together by a simultaneous induction on the complexity of ψ. We omit the details here.

Next we note that **(b)** can be strengthened as follows:

(c) If $\psi \in \mathcal{F}'$, $\mathcal{T} \in W$ and there is no finite $\Gamma \subseteq \mathcal{T}$ such that $\Gamma \Rightarrow \psi$ has an $\mathcal{S}$-cut-free-proof from $\mathcal{S}$, then $v(\mathcal{T}, \psi) = f$.

Indeed, under these conditions $\mathcal{T}$ can be extended to a ψ-maximal theory $\mathcal{T}'$. Now $\mathcal{T}' \in W$, $\mathcal{T} \subseteq \mathcal{T}'$, and by **(b)**, $v(\mathcal{T}', \psi) = f$. Hence also $v(\mathcal{T}, \psi) = f$.

Now **(a)** and **(b)** together imply that $v(\mathcal{T}_0, \psi) = t$ for every $\psi \in \Gamma_0 \subseteq \mathcal{T}_0$, and $v(\mathcal{T}_0, \varphi_0) = f$. Hence $\mathcal{W}$ is not a model of s. We end the proof by showing that $\mathcal{W}$ is a model of $\mathcal{S}$. So let $\psi_1, \ldots, \psi_n \Rightarrow \theta \in \mathcal{S}$ and let $\mathcal{T} \in W$, where $\mathcal{T}$ is φ-maximal. Assume by way of contradiction that $v(\mathcal{T}, \psi_i) = t$ for $1 \leq i \leq n$, while $v(\mathcal{T}, \theta) = f$. By **(c)**, for every $1 \leq i \leq n$ there is a finite $\Gamma_i \subseteq \mathcal{T}$ such that $\Gamma_i \Rightarrow \psi_i$ has an $\mathcal{S}$-cut-free-proof from $\mathcal{S}$. On the other hand $v(\mathcal{T}, \theta) = f$ implies (by **(a)**)

[6] This inductive definition isn't totally formal, since satisfaction by a substitution is defined for a generalized $\mathcal{L}$-frame, which we are in the middle of constructing, but the intention should be clear.

that $\theta \notin \mathcal{T}$. Since $\mathcal{T}$ is φ-maximal, it follows that there is a finite $\Sigma \subseteq \mathcal{T}$ such that $\Sigma, \theta \Rightarrow \varphi$ has an $\mathcal{S}$-cut-free-proof from $\mathcal{S}$. Now from $\Gamma_i \Rightarrow \psi_i$ $(1 \leq i \leq n)$, $\Sigma, \theta \Rightarrow \varphi$, and $\psi_1, \ldots, \psi_n \Rightarrow \theta$ one can infer $\Gamma_1, \ldots, \Gamma_n, \Sigma \Rightarrow \varphi$ by $n+1$ $\mathcal{S}$-cuts (on $\psi_1, \ldots, \psi_n$ and θ). It follows that the last sequent has an $\mathcal{S}$-cut-free-proof from $\mathcal{S}$. Since $\Gamma_1, \ldots, \Gamma_n, \Sigma \subseteq \mathcal{T}$, this contradicts the φ-maximality of $\mathcal{T}$. □

Theorem 4. (**Soundness and Completeness**) Every canonical constructive system **G** is strongly sound and complete with respect to the semantics of **G**-legal generalized frames. In other words:

1. $T \vdash_{\mathbf{G}} \varphi$ iff $T \models_{\mathbf{G}} \varphi$.
2. $\mathcal{S} \vdash_{\mathbf{G}}^{seq} s$ iff $\mathcal{S} \models_{\mathbf{G}}^{seq} s$.

Proof. Immediate from Theorems 3 and 2, and Propositions 1, 3. □

Corollary 1. *If* **G** *is a canonical constructive system in* $\mathcal{L}$ *then* $\langle \mathcal{L}, \models_{\mathbf{G}} \rangle$ *is a logic.*

Corollary 2. (**Compactness**) *Let* **G** *be a canonical constructive system.*

1. If $\mathcal{S} \models_{\mathbf{G}}^{seq} s$ *then there exists a finite* $\mathcal{S}' \subseteq \mathcal{S}$ *such that* $\mathcal{S}' \models_{\mathbf{G}}^{seq} s$.
2. $\models_{\mathbf{G}}$ *is finitary.*

Theorem 5.

1. (**General Strong Cut Elimination Theorem**) *Every canonical constructive system* **G** *admits strong cut-elimination (see Definition 12).*
2. (**General Cut Elimination Theorem**) *A sequent is provable in a canonical constructive system* **G** *iff it has a cut-free proof there.*

Proof. The first part follows from Theorem 4 and Theorem 3. The second part is a special case of the first, where the set $\mathcal{S}$ of premises is empty. □

Corollary 3. *The following conditions are equivalent for a canonical single-conclusion Gentzen-type system* **G***:*

1. $\langle \mathcal{L}, \vdash_{\mathbf{G}} \rangle$ *is a logic (by Proposition 2, this means that* $\vdash_{\mathbf{G}}$ *is consistent).*
2. **G** *is coherent.*
3. **G** *admits strong cut-elimination.*
4. **G** *admits cut-elimination.*

Proof. 1 implies 2 by Theorem 1. 2 implies 3 by Theorem 5. 3 trivially implies 4. Finally, without using cuts there is no way to derive $p_1 \Rightarrow p_2$ in a canonical Gentzen-type system. Hence 4 implies 1. □

5 Analycity and Decidability

In general, in order for a denotational semantics of a propositional logic to be useful and effective, it should be *analytic*. This means that to determine whether a formula φ follows from a theory $\mathcal{T}$, it suffices to consider *partial* valuations, defined on the set of all subformulas of the formulas in $\mathcal{T} \cup \{\varphi\}$. Now we show that the semantics of **G**-legal frames is analytic in this sense.

Definition 19. Let **G** be a canonical constructive system for $\mathcal{L}$. A **G**-legal *semiframe* is a triple $\mathcal{W}' = \langle W, \leq, v' \rangle$ such that:

1. $\langle W, \leq \rangle$ is a nonempty partially ordered set.
2. v' is a partial function from the set of formulas of $\mathcal{L}$ into the set of persistent functions from W into $\{t, f\}$ such that:
   - $\mathcal{F}'$, the domain of v', is closed under subformulas.
   - v' respects the rules of **G** on $\mathcal{F}'$ (e.g.: if r is an introduction rule for an n-ary connective $\diamond$, and $\diamond(\psi_1, \dots, \psi_n) \in \mathcal{F}'$, then $v(a, \diamond(\psi_1, \dots, \psi_n)) = t$ whenever all the premises of r are satisfied in a by a substitution σ such that $\sigma(p_i) = \psi_i$ $(1 \leq i \leq n)$).

Theorem 6. *Let* **G** *be a canonical constructive system for* $\mathcal{L}$. *Then the semantics of* **G***-legal frames is* analytic *in the following sense: If* $\mathcal{W}' = \langle W, \leq, v' \rangle$ *is a* **G***-legal* semiframe*, then* v' *can be extended to a function* v *so that* $\mathcal{W} = \langle W, \leq, v \rangle$ *is a* **G***-legal frame.*

Proof. Let $\mathcal{W}' = \langle W, \leq, v' \rangle$ be a **G**-legal semiframe. We recursively extend v' to a total function v. For atomic p we let $v(p) = v'(p)$ if $v'(p)$ is defined, and $v(p) = \lambda a \in W.t$ (say) otherwise. For $\varphi = \diamond(\psi_1, \dots, \psi_n)$ we let $v(\varphi) = v'(\varphi)$ whenever $v'(\varphi)$ is defined, and otherwise we define $v(\varphi, a) = f$ iff there exists an elimination rule r with $\diamond(p_1, \dots, p_n) \Rightarrow$ as its conclusion, and an element $b \geq a$ of W, such that all premises of r are satisfied in b (with respect to $\langle W, \leq, v \rangle$) by a substitution σ such that $\sigma(p_j) = \psi_j$ $(1 \leq j \leq n)$. Note that the satisfaction of the premises of r by σ in elements of W depends only on the values assigned by v to $\psi_1, \dots, \psi_n$, so the recursion works, and v is well defined. From the definition of v and the assumption that $\mathcal{W}'$ is a **G**-legal semiframe, it immediately follows that v is an extension of v', that $v(\varphi)$ is a persistent function for every φ (so $\mathcal{W} = \langle W, \leq, v \rangle$ is a generalized $\mathcal{L}$-frame), and that $\mathcal{W}$ respects all the elimination rules of **G**. Hence it only remains to prove that it respects also the introduction rules of **G**. Let $r = \{\Pi_i \Rightarrow q_i\}_{1 \leq i \leq m} / \ \Rightarrow \diamond(p_1, p_2, \dots, p_n)$ be such a rule, and assume that for every $1 \leq i \leq m$, $\sigma(\Pi_i) \Rightarrow \sigma(q_i)$ is true in a with respect to $\langle W, \leq, v \rangle$. We should show that $v(a, \diamond(\psi_1, \dots, \psi_n)) = t$.

If $v'(a, \diamond(\psi_1, \dots, \psi_n))$ is defined, then since its domain is closed under subformulas, for every $1 \leq i \leq n$ and every $b \in W$ $v'(b, \psi_i)$ is defined. In this case, our construction ensures that for every $1 \leq i \leq n$ and every $b \in W$ we have $v'(b, \psi_i) = v(b, \psi_i)$. Therefore, since for every $1 \leq i \leq m$, $\sigma(\Pi_i) \Rightarrow \sigma(q_i)$ is locally true in every $b \geq a$ with respect to $\langle W, \leq, v \rangle$, it is also locally true with respect to $\langle W, \leq, v' \rangle$. Since v' respects r, $v'(a, \diamond(\psi_1, \dots, \psi_n)) = t$, so $v(a, \diamond(\psi_1, \dots, \psi_n)) = t$ as well, as required.

Now, assume $v'(a, \diamond(\psi_1, \ldots, \psi_n))$ is not defined, and assume by way of contradiction that $v(a, \diamond(\psi_1, \ldots, \psi_n)) = f$. So, there exists $b \geq a$ and an elimination rule $\{\Delta_j \Rightarrow \Sigma_j\}_{1 \leq j \leq k} / \ \diamond(p_1, p_2, \ldots, p_n) \Rightarrow$ such that $\sigma(\Delta_j) \Rightarrow \sigma(\Sigma_j)$ is locally true in b for $1 \leq j \leq k$. Since $b \geq a$, our assumption about a implies that $\sigma(\Pi_i) \Rightarrow \sigma(q_i)$ is locally true in b for $1 \leq i \leq m$. It follows that by defining $u(p) = v(b, \sigma(p))$ we get a valuation u in $\{t, f\}$ which satisfies all the clauses in the union of $\{\Pi_i \Rightarrow q_i \mid 1 \leq i \leq m\}$ and $\{\Delta_j \Rightarrow \Sigma_j \mid 1 \leq j \leq k\}$. This contradicts the coherence of **G**. □

The following two theorems are now easy consequence of Theorem 6 and the soundness and completeness theorems of the previous section:[7]

Theorem 7. *Let* **G** *be a canonical constructive system. Then* **G** *is strongly decidable: Given a finite set $\mathcal{S}$ of sequents, and a sequent s, it is decidable whether $\mathcal{S} \vdash^{seq}_{\mathbf{G}} s$ or not. In particular: it is decidable whether $\Gamma \vdash_{\mathbf{G}} \varphi$, where φ is formula and Γ is a finite set of formulas.*

Proof. Let $\mathcal{F}'$ be the set of subformulas of the formulas in $\mathcal{S} \cup \{s\}$. From Theorem 6 and the proof of Theorem 3 it easily follows that in order to decide whether $\mathcal{S} \vdash^{seq}_{\mathbf{G}} s$ it suffices to check all triples of the form $\langle W, \subseteq, v' \rangle$ where $W \subseteq 2^{\mathcal{F}'}$ and $v' : \mathcal{F}' \rightarrow (W \rightarrow \{t, f\})$, and see if any of them is a **G**-legal semiframe which is a model of $\mathcal{S}$ but not a model of s. □

Theorem 8. *Let* $\mathbf{G_1}$ *be a canonical constructive system in a language $\mathcal{L}_1$, and let* $\mathbf{G_2}$ *be a canonical constructive system in a language $\mathcal{L}_2$. Assume that $\mathcal{L}_2$ is an extension of $\mathcal{L}_1$ by some set of connectives, and that* $\mathbf{G_2}$ *is obtained from* $\mathbf{G_1}$ *by adding to the latter canonical rules for connectives in $\mathcal{L}_2 - \mathcal{L}_1$. Then* $\mathbf{G_2}$ *is a conservative extension of* $\mathbf{G_1}$ *(i.e.: if all formulas in $\mathcal{T} \cup \{\varphi\}$ are in $\mathcal{L}_1$ then $\mathcal{T} \vdash_{\mathbf{G_1}} \varphi$ iff $\mathcal{T} \vdash_{\mathbf{G_2}} \varphi$).*

Proof. Suppose that $\mathcal{T} \not\vdash_{\mathbf{G_1}} \varphi$. Then there is $\mathbf{G_1}$-legal model $\mathcal{W}$ of $\mathcal{T}$ which is not a model of φ. Since the set of formulas of $\mathcal{L}_1$ is a subset of the set of formulas of $\mathcal{L}_2$ which is closed under subformulas, Theorem 6 implies that $\mathcal{W}$ can be extended to a $\mathbf{G_2}$-legal model of $\mathcal{T}$ which is not a model of φ. Hence $\mathcal{T} \not\vdash_{\mathbf{G_2}} \varphi$. □

Note. In [7] (his famous response to [13]), Belnap suggested that the rules for a connective $\diamond$ should be *conservative*, in the sense that if $\mathcal{T} \vdash \varphi$ is derivable using them, and $\diamond$ does not occur in $\mathcal{T} \cup \varphi$, then $\mathcal{T} \vdash \varphi$ can also be derived without using the rules for $\diamond$. Now our notion of coherence provides an effective necessary and sufficient criterion for checking whether a given set of canonical rules is conservative in this sense. Moreover: Theorem 8 shows that a very strong form of Belnap's conservativity criterion is valid for canonical constructive systems, and so what a set of canonical rules defines is system-independent.

[7] The two theorems can also be proved directly from the cut-elimination theorem for canonical constructive systems. We leave this to the full paper.

6 Related and Further Works

There have been several works in the past on conditions for cut-elimination. Except for [6], the closest to the present one is [8]. The range of systems dealt with there is in fact broader than ours, since it deals with various types of structural rules, while in this paper we assume the standard structural rules of minimal logic. On the other hand, our coherence criterion is much simpler than the reductivity criterion of [8], while our strong cut-elimination is stronger then the reductive cut-elimination of [8]. Another crucial similarity is that both papers use nondeterministic semantic frameworks (in [8] this is only implicit). However, while we use the concrete framework of intuitionistic-like Kripke frames, variants of the significantly more abstract phase semantics are used in [8].

Another difference is that unlike the present work, [8] treats also systems which allow the use in derivations of negative sequents. Our next task is to extend our framework and results so they apply to systems of this sort as well.

References

1. Avron, A.: Simple Consequence Relations. Information and Computation 92, 105–139 (1991)
2. Avron, A.: Gentzen-Type Systems, Resolution and Tableaux. Journal of Automated Reasoning 10, 265–281 (1993)
3. Avron, A.: A Nondeterministic View on Nonclassical Negations. Studia Logica 80, 159–194 (2005)
4. Avron, A.: Non-deterministic Semantics for Families of Paraconsistent Logics. In: Beziau, J.-Y., Carnielli, W., Gabbay, D.M. (eds.) Handbook of Paraconsistency. Studies in Logic, vol. 9, pp. 285–320. College Publications (2007)
5. Avron, A., Lev, I.: Canonical Propositional Gentzen-Type Systems. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 529–544. Springer, Heidelberg (2001)
6. Avron, A., Lev, I.: Non-deterministic Multiple-valued Structures. Journal of Logic and Computation 15, 24–261 (2005)
7. Belnap, N.D.: Tonk, Plonk and Plink. Analysis 22, 130–134 (1962)
8. Ciabattoni, A., Terui, K.: Towards a Semantic Characterization of Cut-Elimination. Studia Logica 82, 95–119 (2006)
9. Fernandez, D.: Non-deterministic Semantics for Dynamic Topological Logic. Annals of Pure and Applied Logic 157, 110–121 (2009)
10. Gentzen, G.: Investigations into Logical Deduction. In: Szabo, M.E. (ed.) The Collected Works of Gerhard Gentzen, pp. 68–131. North Holland, Amsterdam (1969)
11. Gurevich, Y., Neeman, I.: The Logic of Infons, Microsoft Research Tech Report MSR-TR-2009-10 (January 2009)
12. Kripke, S.: Semantical Analysis of Intuitionistic Logic I. In: Crossly, J., Dummett, M. (eds.) Formal Systems and Recursive Functions, pp. 92–129. North-Holland, Amsterdam (1965)
13. Prior, A.N.: The Runabout Inference Ticket. Analysis 21, 38–39 (1960)
14. Sundholm, G.: Proof theory and Meaning. In: Gabbay, D.M., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. 9, pp. 165–198 (2002)

A Novel Architecture for Situation Awareness Systems

Franz Baader[1], Andreas Bauer[2,3], Peter Baumgartner[2,3],
Anne Cregan[3], Alfredo Gabaldon[4], Krystian Ji[3], Kevin Lee[3,5],
David Rajaratnam[3,5], and Rolf Schwitter[6]

[1] Technische Universität Dresden, Germany
baader@tcs.inf.tu-dresden.de
[2] Australian National University
Firstname.Lastname@anu.edu.au
[3] National ICT Australia (NICTA*), Australia
Firstname.Lastname@nicta.com.au
[4] New University of Lisbon, Portugal, Center for AI
ag@di.fct.unl.pt
[5] University of New South Wales, Australia
[6] Macquarie University, Australia
rolfs@ics.mq.edu.au

Abstract. *Situation Awareness* (SA) is the problem of comprehending elements of an environment within a volume of time and space. It is a crucial factor in decision-making in dynamic environments. Current SA systems support the collection, filtering and presentation of data from different sources very well, and typically also some form of *low-level* data fusion and analysis, e.g., recognizing patterns over time. However, a still open research challenge is to build systems that support *higher-level* information fusion, viz., to integrate domain specific knowledge and automatically draw conclusions that would otherwise remain hidden or would have to be drawn by a human operator. To address this challenge, we have developed a novel system architecture that emphasizes the rôle of formal logic and automated theorem provers in its main components. Additionally, it features controlled natural language for operator I/O. It offers three logical languages to adequately model different aspects of the domain. This allows to build SA systems in a more declarative way than is possible with current approaches. From an automated reasoning perspective, the main challenges lay in combining (existing) automated reasoning techniques, from low-level data fusion of time-stamped data to semantic analysis and alert generation that is based on linear temporal logic. The system has been implemented and interfaces with Google-Earth to visualize the dynamics of situations and system output. It has been successfully tested on realistic data, but in this paper we focus on the system architecture and in particular on the interplay of the different reasoning components.

1 Introduction

Situation Awareness (SA from now on) is concerned with the perception of elements in the environment within a volume of time and space, the comprehension of their meaning

* NICTA is funded by the Australian Government's *Backing Australia's Ability* initiative.

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 77–92, 2009.

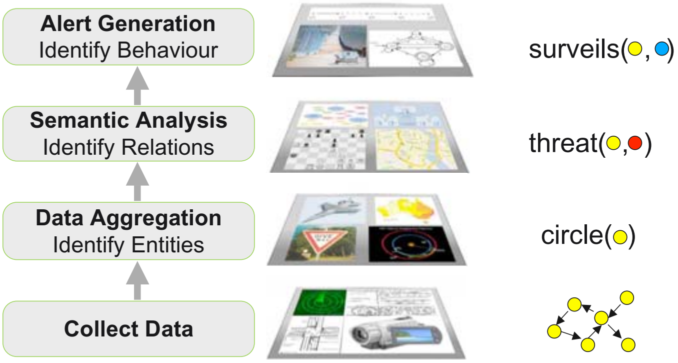


Fig. 1. SAIL high-level system architecture

and the projection of their status in the near future [End95]. Having complete, accurate and current SA is highly desirable for managing real-time dynamic systems including military and emergency scenarios, air traffic control and other transport networks. Poor SA is a key contributor to critical human errors, usually tracing back to cognitive overload or poor information transfer between operators.

According to Endsley's model [End95], the levels of SA are perception, comprehension and projection. Currently, the challenge of integrating heterogeneous information into a single composite picture of the environment at the semantic level of human comprehension and projection remains open. To address this challenge, we have developed a novel system architecture and implemented a system as a part of the *Situation Awareness by Inference and Logic* (SAIL) project. It provides functionality in three successive tiers (Fig. 1) corresponding roughly to Endsley's levels of SA.

Data Aggregation involves monitoring data sources (e.g. track data obtained from radar) to identify entities of a situation and their low-level properties (e.g. that a trajectory contains a circle). This corresponds to the perception level of SA.

Semantic Analysis involves interpretation and evaluation of the entities in conjunction with background knowledge, producing an understanding of the overall meaning of the identified entities: how they relate to each other, what kind of situation it is, what it means in terms of one's mission goals (e.g. that a fighter plane is threatening some object). This corresponds to the comprehension level of SA.

Alert Generation involves monitoring and projecting how events may unfold over time. Based on the interpretation of a situation, this functionality identifies possible evolutions of the current situation (e.g. of an aircraft currently surveilling a border). If a potentially high-impact situation is recognised to arise, an alert is sent to the operator. This relates to the projection level of SA.

We emphasize the role of declarative techniques in all three layers, each one realized essentially by specification in a formal logic. This bears the advantage that we have a precise semantics for each layer, defined in terms of formal logic rather than the implemented behaviour of a specific set of tools. As formal semantics can be captured in an abstract manner, our approach does not tie users to specific implementations. However, to make the employed logics operational, we capitalize on the state of the art by utilizing two of the latest available (tableau-based) reasoners, E-KRHyper [PW07] and RacerPro [HM03]. We exploit E-KRHyper's model-building abilities to deliver its result as a description-logic ABox to RacerPro. One caveat here lies in the dynamic nature of our application, which requires reasoning about data that changes over time. Unfortunately, the currently available (first-order and description logic) reasoners do not offer suitable services, so we solve this problem via a novel architecture, the *SAIL* architecture, that utilizes a control component to invoke the reasoners in specific ways. Other important aspects of SAIL concern the use of controlled natural language (CNL) as the primary means of interaction with human operators, the integration of a public-domain GIS system into E-KRHyper for basic geometrical calculations. The architecture has been implemented in the *SAIL system* and tested successfully on realistic data. Although we refer occasionally to the system, the focus of this paper is on its architecture.

1.1 Related Work

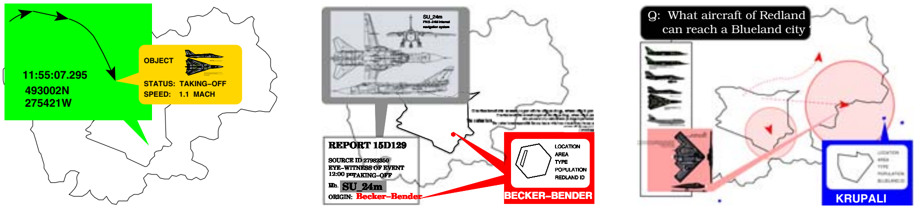
Several systems for SA have been developed that support the management of various information sources (sensor data, textual information, databases, etc.) for purposes such as information exchange and graphical presentation to facilitate decision making. Information retrieval techniques to filter and rank a potentially overwhelming stream of information are often applied to facilitate this process. Typical examples of academic prototypes, from the military domain, are [GHGJ+07, SRS+07]. However, such systems lack capabilities that enable a deep, semantical modelling of the domain and drawing conclusions on top of it. Therefore, information integration to assess and project a situation into the future still has to occur largely in the "heads of the decision makers", which leads to the *Semantic Challenge* in SA [NL05]. These issues are a recurring theme in current SA research and attract considerable attention worldwide (see a recent overview paper by domain experts [BKS+06]). It is not surprising that (in particular) *description logics* [BCM+03] and corresponding reasoners have attracted a lot of attention in the SA community. Although some SA tools, such as [SRS+07, GHGJ+07] reportedly make use of description logic reasoners, it becomes clear that deep, semantic reasoning is not yet implemented in a satisfying manner. Moreover, the cited reports make little mention of the lower levels of the information fusion process; there is an implicit, underlying assumption that low-level data, such as that provided by physical sensors, is already in a semantically and syntactically well structured form, associating to real-world events the concrete objects, the time of occurrence, and data sources, which may be stored in a database or ontology. However, how to actually obtain this meta information is not detailed upon (cf. [MKL+05]). To the best of our knowledge, no prior SA system has managed to use an integrated semantic approach based on logical inference and reasoning, from the sensor-data level to the higher levels of situation assessment, and beyond (e.g., impact assessment, projection into the future, etc.). This is one of the key differences between our proposed system, whose functioning can be described largely based on formal logic, and the existing ones, which combine ad-hoc representations with formal reasoning.

1.2 Running Example

We will use the scenario depicted in Fig. 2 as a running example to illustrate the various components of the SAIL architecture. Our work is embedded in a project run in collaboration with the Defence Science and Technology Organisation (DSTO), Australia. The scenario discussed here is a small excerpt from a scenario developed by DSTO and NATO partners.

Suppose we are interested in monitoring a geographical region for potentially hostile activity by a neighbouring country. Our system receives low-level track data from electronic surveillance systems (e.g. radar) that combined with Geographic Information System (GIS) data allow us to keep track of the movement of vehicles in the region. In this scenario, radar data indicates that an aircraft takes off at 11:55 and travels at a speed of 1.1 Mach (Fig. 2a). Another source of information to the system are natural language reports from human witnesses in the region. These reports frequently contain information that the other sources cannot deliver, such as the specific type of aircraft. In our scenario, such a report indicates that an SU_24M was seen taking off from Becker-Bender at 12:00.

Interacting with the system is a user issuing queries in (Controlled) Natural Language. In this example, a submitted query is “What aircraft of Redland can reach a city of Blueland?” (Fig. 2c). The SAIL system 1) uses its various sources of information to speculate that the aircraft detected by radar and the aircraft described in the report are the same; 2) uses what it knows about the detected SU_24M and general background knowledge, such as aircraft capabilities (e.g. travel distance range), geographical information, and the types of aircraft used by different countries, to determine that the detected SU_24M is Redland’s and that it is currently capable of reaching a city of Blueland, and 3) includes it in the answer to the user’s query. Moreover, in this example the SU_24M and any other aircraft that have been detected are monitored for *aggressive* behavior. If an aircraft is determined to be aggressive, the system issues an alert.



(a) Surveillance systems (e.g. radar) detect an aircraft taking off at 11:55.

(b) A witness reports seeing an SU_24M taking-off from Becker-Bender at 12:00.

(c) A SAIL-system user submits the CNL query “What aircraft of Redland can reach a city of Blueland?”

Fig. 2. Example Scenario: multiple sources of information are fused and analysed in the detection of a potentially hostile aircraft taking off

2 SAIL Architecture

Fig. 3 depicts the SAIL system architecture. SAIL takes two kinds of input: *data streams* and *eye-witness reports*. Data streams provide time-stamped sensor data about aircraft, ships, etc, giving their location, speed, acceleration and so on, shown as the boxes $\mathsf{SD}_i, \mathsf{SD}_{i+1}, \mathsf{SD}_{i+2}, \ldots$. In our scenario, new data arrives about every 0.33 seconds and concerns about 30 objects. SAIL accumulates this information: at each time point i the system stores data from previous time points SD_j, for $j \leq i$. For practical reasons – to cope with the amount of data accumulating over time – SAIL supports a user-configurable *time window* and abandons data time-stamped prior to that window. Eye-witness reports are represented in a time-stamped relational way, similarly to the sensor data. As they are originally expressed in a form of controlled natural language (CNL), some preprocessing is needed to arrive at a relational form (cf. Sec. 2.4). The control program presented in Algorithm 1 coordinates the processing of all that.

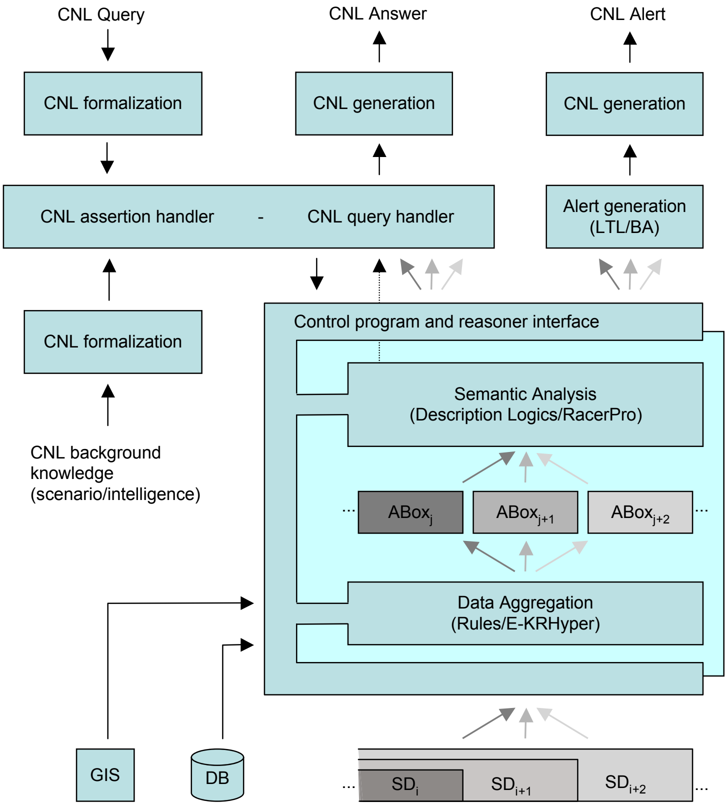

CNL Query
CNL Answer
CNL Alert
CNL formalization
CNL generation
CNL generation
CNL assertion handler - CNL query handler
Alert generation (LTL/BA)
CNL formalization
CNL background knowledge (scenario/intelligence)
Control program and reasoner interface
Semantic Analysis (Description Logics/RacerPro)
... ABox$_j$ ABox$_{j+1}$ ABox$_{j+2}$ ...
Data Aggregation (Rules/E-KRHyper)
GIS
DB
... SD$_i$ SD$_{i+1}$ SD$_{i+2}$ ...

Fig. 3. SAIL system architecture. Abbreviations: GIS - GIS system, SD_i - sensor data, LTL - linear temporal logic, BA - Büchi automaton, CNL - controlled natural language.

```
input: a TBox $tbox$, window size $n$
$t \leftarrow 0$;
for $i \leftarrow 0$ to $n-1$ do                /* initialize sensor data windows */
    $sd[i] \leftarrow \emptyset$;
end
while $true$ do
    $sd[t \bmod n] \leftarrow$ await input data from CNL assertion handler;
    $user\_queries \leftarrow$ read nRQL queries from CNL query handler;
    $SD_t \leftarrow \bigcup_{i=0}^{n-1} sd[i]$;
    $ABox_t \leftarrow$ InvokeDAModule($SD_t$);
    $tptp\_answer \leftarrow$ InvokeSAModule($tbox, ABox_t, user\_queries$);
    $data\_trace \leftarrow$ InvokeSAModule($tbox, ABox_t$);
    send $tptp\_answer$ to CNL query handler;
    send $data\_trace$ to alert generation module;
    $t \leftarrow t+1$;
end
```

Algorithm 1. Control procedure for query answering and data trace generation

The control procedure takes two arguments - a TBox *tbox* and a configurable window size n. There are two arrays *sd* and *data_trace*. Each array is of size n and each element in the array is initially set to an empty set. The main part of the procedure consists of a while-loop that runs indefinitely. In each iteration, the procedure is blocked until the next input arrives from the CNL assertion handler. It then proceeds by reading *user_queries* in nRQL format from the query handler. Positions of array *sd* are filled with sensor data in sequence from index 0 to $n-1$ as t increments with each iteration. It simply loops back to position $(t \bmod n)$ if $t > n$, thus disregarding input data collected at time point $t-n$. That is, we retain at most n time points of input data prior to the current time point t.

Collected input data in the array *sd* are aggregated to form SD_t. This data, together with GIS and database information, are processed by the data aggregation module via `InvokeDAModule`, to produce an ABox $ABox_t$ marked with the current time point t. This new ABox is loaded together with the input TBox *tbox* to the semantic analysis module via `InvokeSAModule` for further processing, where user queries *user_queries* will be executed. The answer produced will be in TPTP format and is immediately sent to the CNL query handler for conversion to CNL output. Similarly, a data trace will be generated via `InvokeSAModule` for time point t which will be sent to the alert generation module for further processing (as described in Sec. 2.3 below).

2.1 Data Aggregation

Data aggregation is the process of gathering information and expressing it in a summary form. With a wide enough time window, this allows to detect object properties over time, like, for instance, to detect a "circle" in an object's trajectory, or that a certain object re-visits a certain point again and again. Eye-witness reports, initially expressed in CNL, also feed into this layer. For instance, a CNL sentence like "An

SU_24M started from Eaglevista at 09:00" could be used to enrich information about a previously unidentified object that it is an SU_24M.

The core component for data aggregation is the theorem prover E-KRHyper. E-KRHyper is sound and complete for first-order logic with equality. It is an implementation of the E-hyper tableau calculus [BFP07]. E-KRHyper has been described in more detail in [PW07].

E-KRHyper is invoked by the control program whenever the time window moves or a new eye-witness report comes in. It then updates the data aggregated so far based on the new information. E-KRHyper accepts if-then rules, of which first-order clause logic is a special case.[1] Rules are used to specify data aggregation in a declarative way. Here are some examples, relevant to the event in Fig. 2a:

```
object_appears(Obj, Now) :- % An Object appears at the current time, Now.
    current_time(Now),      % Current time - supplied by control program
    object(Obj, Now),       % Get some Object existing at Now
    previous_time(Now, T),  % Previous time - supplied by control program
    \+ object(Obj, T).      % Check that Object was not there at T

take_off(Event, Obj, Now) :- % a new Event: an Object takes off Now
    object_appears(Obj, Now),
    in_air(Obj, Now),        % in_air computed by GIS
    concat(['ev_',Obj,'_',Now],Event). % creates unique Event id
```

The rules are applied in a bottom-up way to the sensor data and eye-witness reports (after conversion into logical form by the CNL assertion handler) until a fixed point is reached. A certain subset of the fixpoint is then extracted and passed on to the semantic analysis layer.

To see how this works, assume that the appearance of the aircraft in Fig. 2a is represented as `object(ac1, '11:55')`, and assume that `in_air(ac1, '11:55')` can be established with the help of the GIS (see below) from the radar data. E-KRHyper then derives the result `take_off('ev_ac1_11:55', ac1, '11:55')`,[2] which is passed on to the semantic analysis layer with the help of rules with a head predicate "`abox`":

```
abox(take_off(Event)) :- take_off(Event, Obj, Time).
```

Such rules specify concept assertions (like `take_off('ev_ac1_11:55')`) or role assertions (like `event_time('ev_ac1_11:55', '11:55')` in the sense of description logics.[3] As indicated in Fig. 3 by using indices *j* rather than *i*, the sequence of ABoxes

[1] The syntax is similar to Prolog, but extends it by a logical-or operator "`;`" which means disjunction in the head of rules. E-KRHyper also supports stratified negation "\+" and evaluation of arithmetic goals. These features are helpful for computations on timepoints and spatial distances. They are used locally inside the data aggregation layer and do not interfere with the overall logic.

[2] We use integers to represent time and writing time points as in `'11:55'` is just for readability here. Names like `'ev_ac1_11:55'` are obtained with a concat-atoms like built-in function.

[3] A collection of such assertions, an ABox, specifies a theory in the sense of first-order logic. Like any first-order theory an ABox may be incomplete, that is, it might entail a given first-order sentence, like e.g. a concept instance `is_AWACS(o1)`, entail its negation, or neither of them.

$\mathsf{ABox}_j, \mathsf{ABox}_{j+1}, \ldots$ does not necessarily correspond to the time points of the sensor data. This is, because it is neither necessary nor feasible to update the data aggregation with the same rate as the sensor data come in. By default, a new ABox is computed every second or when a new eye-witness report comes in.

Preserving information over time. Each ABox $\mathsf{ABox}_j, \mathsf{ABox}_{j+1}, \ldots$ represents information for the time point "now". Would ABoxes include their predecessors (as is the case with the sensor data) then a massive frame problem is to be expected. However, ABox assertions that are consistent with all later ABoxes can be kept without problems. For instance, it might be useful to keep the "take off" event synthesized in the example above until its object no longer exists. This can be realized by writing rules with special predicate symbol reassert in the head:

```
reassert(take_off(Event, Obj, CreationTime)) :-
     take_off(Event, Obj, CreationTime), current_time(Time),
     object(Obj, Time).  % reassert as long as its object exists
```

Like the abox predicate, the reassert predicate is interpreted by the control program in a special way, by just feeding its extension in the current computed model into the next invocation of E-KRHyper. This achieves the desired effect.

The formal meaning of this type of data aggregation can be given in a similar way as for production system languages (cf. [Ras94]) and E-KRHyper merely serves as an implementation of that. However, as pointed out above, this particular choice is not mandatory. We have chosen E-KRHyper because of our familiarity with the system. The use of equality reasoning is not crucial, but the ability to generate models in a bottom-up manner is, in order to extract ABoxes from the computed models. Another important detail is that the rules we use fall into a formula class that E-KRHyper is a decision procedure for.

Databases and GIS. SAIL's data aggregation layer also integrates databases and a public-domain GIS. The (relational) databases here contain data about capabilities of aircraft, ships, etc, such as maximum speed, range, and so on. As they are rather small, instead of coupling a proper DBMS, we simply expressed the knowledge base directly in E-KRHyper's input language, as a set of facts.

The sensor data consists of spatial information about objects that are moving over time. The computation of their properties will often involve spatial computations in the GIS, numerical computations and database lookups. A simple example is to compute whether a certain aircraft can reach a certain destination. This requires a database query of the aircraft's range, the aircraft's time in the air, and the distance to the destination. For that, it is useful to be able to compute basic spatial properties of objects, such as whether an object is in air, within a certain region (for example, a country) or targets some city. To this end, we have integrated the popular open source GDAL/OGR GIS library into the SAIL data aggregation layer. The GIS utilises vector data about geographic features, described in terms of geometric objects such as points, lines, and polygons. The interface to the GIS is realized by special predicates and functions, which can be used in rule bodies to invoke its services.

For example, determining whether a city can be reached by a given aircraft can be performed by using a combination of builtins; first to define the geographic region that the aircraft can reach, and then to test whether the city is within that region:

```
reachableCity(Aircraft, Range, City) :-
    Reach is ogr_g_buffer(Aircraft, Range),
    ogr_g_within(City, Reach).
```

2.2 Semantic Analysis

The semantic analysis layer is where situation assessment occurs. This component of the system utilizes a logical description of domain properties at a conceptually higher-level than what the data aggregation layer produces. While the data aggregation layer generates assertions about, e.g., the location of objects, the semantic analysis layer attaches higher level meaning to the situation, e.g. whether the object is behaving aggressively. In addition to the information coming from the data aggregation layer, the semantic analysis layer has at its disposal a background knowledge base (an ontology) containing information about the different aircraft, ships, and other vehicle types available to various countries, the capabilities of the vehicles, e.g. weapons, travel range; the status of the relationship between countries, e.g. ally, neutral, hostile; and other similar domain background knowledge. The knowledge base also contains a collection of definitions of *events*, e.g., move, fly, sail, depart, take_off, etc, that is in part inspired by the event-semantics used in the NLP community. From this knowledge base and the information delivered by the data aggregation layer, the semantic analysis layer computes, by logical reasoning, a deeper, more concise high-level description of the current situation. In the following we give only a flavor of the concrete knowledge base with a small example around "aggressive behavior".

The particular representation and reasoning formalism used in this layer is description logics (DL) [BCM+03]. A typical DL knowledge base has two components: an ABox and a TBox. ABox assertions are of the form $C(x)$ or $R(x,y)$, where C and R denote *concept* and *role* respectively, and x,y are individuals. Assertions are provided to the semantic analysis layer as background information, as eye-witness reports or generated by the data aggregation layer. For example, the assertions below state two facts - $t1$ being a physical object and $t1$ being a target of another object $o1$:

$$physical\_object(t1) \qquad has\_target(o1,t1)$$

TBox axioms are of the form $C \sqsubseteq D$ or $C \doteq D$. The former requires that every individual belonging to C also belongs to D. The latter equivalence axiom requires that both $C \sqsubseteq D$ and $D \sqsubseteq C$ hold. TBox axioms are pre-defined in the semantic analysis layer. They are combined with the ABoxes from the data aggregation layer to draw additional inferences.

For example, the following axiom defines an aggressive object as an individual that has a target of either a physical object or a space region:

$$aggressive \doteq \exists has\_target.(physical\_object \sqcup space\_region)$$

It then follows that $o1$ is an aggressive object, $\mathit{aggressive}(o1)$. Note that the axiom above is defined as an equivalence, which means we are not only interested in the sufficient condition for classifying an object as aggressive, but the sufficient and necessary conditions. For example, suppose we discover the fact $\mathit{aggressive}(o1)$ from an eye-witness. The above axiom would infer that there is *some* unnamed object which is a target of $o1$ and is either a physical object or a space region.

Also related to "aggressive" is the thematic role *has_target*, which is used to represent the target in an aggression event. Like the concept *aggressive*, this role is also non-primitive and is similarly defined in this layer in terms of other primitive concepts and roles. Specifically, non-primitive roles are defined by means of rules in the DL system language. For instance, the following rule defines *has_target*:

```
(firerule (and (?EM move) (?EM ?Ag has_theme) (?Ag fighter)
          (?Ag ?Org associated_with) (?Org s_blueland enemy_organization)
          (?EM ?Y has_direction) (?Y s_blueland associated_with))
   ((related (new-ind aggr ?Ag ?Y) ?Y has_target)))
```

This rule can be read as follows: if there is a *move* event whose theme (agent), `?Ag`, is a fighter aircraft associated with an enemy organization of Blueland and `?Ag` is moving towards `?Y` (a physical object or a space region) which is associated with Blueland, then this agent `?Ag` has `?Y` as a target of aggression.

The connection between the data aggregation layer and the semantic layer is achieved through primitive concepts/roles. Primitive concepts and roles are not fully defined in DL, but are continuously populated by instances computed by the data aggregation layer. The implementation of the rules filling the primitives needs to be sound wrt. their intended meaning. Completeness, however, is not required and is generally impossible to achieve (i.e., limited possibilities of observing the world). The DL reasoner can then also deduce assertions involving defined concepts. The eye-witness reports may directly yield assertions for defined concepts.

Implementation. Our implementation of the semantic analysis module utilizes RacerPro [HM03]. RacerPro is invoked by the same control program that drives the data aggregation layer. With RacerPro running in server mode, the control program loads the TBox (i.e. the ontology) containing axioms defining high-level concepts like *aggressive* and an ABox containing static background knowledge, then it enters into a loop. In each iteration of the loop, the control program loads into RacerPro the most recent ABox produced by the data aggregation layer and then executes a number of DL rules, such as the one above, that essentially extend the ABox with additional, inferred facts. Once this is done, the reasoner has a full knowledge base and is ready for query processing. Three classes of queries are issued to the reasoner: 1) Localization queries, which are automatically issued by the control program and whose answer is used to display the various objects currently being tracked on a Google-Earth interface. 2) User queries, issued in CNL as described in Sec. 2.4 below. (It is also possible for a user to pose queries directly in the language of the reasoner.) 3) Alert queries, which conceptually belong to the Alert layer but are realized by description logic reasoning (See Sec. 2.3).

2.3 Alerts

In contrast to queries, whose answering is triggered by questions submitted by the user, alerts are raised automatically by the SAIL system. For instance, the user may want to be notified by the system whenever an aircraft crosses a predefined border or an air corridor. In general, an alert describes a critical situation, whose occurrence should be pointed out to the user immediately, without requiring additional interaction. An alert can be created by formally specifying the critical situation in linear time temporal logic (LTL) [Pnu77]. The reason for using *temporal* logic is that the occurrence of a critical situation usually depends on the dynamic behaviour of objects. Single time points are described by the ABoxes generated by the data aggregation layer of SAIL and extended by the semantic analysis layer. These form the atomic propositions in the LTL formulas, i.e., statements about the properties of named objects formulated in terms of the concepts and roles occurring in the ontology. Once an alert is formally specified, a *monitor* is created, which, based on the observations so far, decides whether or not the alert should be raised.

From a formal point of view, an LTL formula φ specifying an alert defines a set L_φ of infinite "words", where each letter in such a word can be seen as a description of the actual state at a given time point. These words correspond to "good" behaviour, i.e., the alert must be raised if the observed sequence of state descriptions does not belong to L_φ. To be more precise, at any given time point, we have only observed a finite prefix of such an infinite sequence. Such a prefix is "bad" if it cannot be extended to an infinite sequence that belongs to L_φ, i.e., however the future behaviour looks like, we know that it cannot become "good." In this case, the alert must be raised. Conversely, the prefix is "good" if all extensions belong to L_φ. In this case, the system no longer needs a monitor for this alert. If none of this is the case, then the system must continue monitoring.

Following an approach developed in the area of runtime verification [BLS06], the monitor for an alert specified by φ is a finite state machine (FSM) with output ("alert", "shut down", "continue"), which can be constructed from the Büchi automaton corresponding to φ (i.e., accepting L_φ).

As an example of an alert specification, consider the following critical situation. If we detect that an enemy aircraft has taken off, and if this aircraft crosses our border, an alarm signal should be raised. The following LTL formula is used to express this:

$$\mathbf{G}(\mathit{in\_air}(p) \Rightarrow \neg \mathit{cross\_border}(p)\ \mathbf{U}\ \mathit{landed}(p)).$$

The temporal operator $\mathbf{G}$ asserts that the formula following it should hold at all future time points, and the until-operator $\mathbf{U}$ asserts that the event $\mathit{cross\_border}(p)$ does not happen before $\mathit{landed}(p)$ is observed. Note that this formula is parametrised with an object name p. The idea is that it is instantiated by all the named objects that are *in_air*. In our running example, alerts are illustrated by monitoring for aggressive events with the simple specification $\mathbf{G}(\neg \mathit{aggressive}(e))$.

Moreover, in our application, for all aircraft p, we keep track of the values of, say, $\mathit{in\_air}(p)$ such that we can, essentially, revert to a propositional representation of the formula. From that, we automatically generate a FSM that reads a trace, which consists of the different truth values of the propositions over time (and obtained via description logic reasoning), and returns in each state whether so far a good prefix was

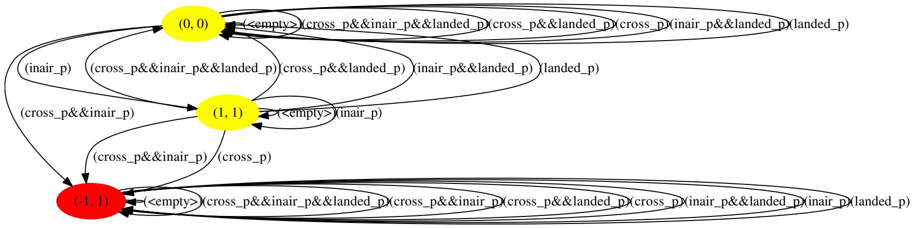


Fig. 4. FSM for $\mathbf{G}(in\_air(p) \Rightarrow \neg cross\_border(p)\ \mathbf{U}\ landed(p))$

observed (i.e., "shut down"), a bad one (i.e., "alert"), or neither (i.e., "continue"). Depending on the formula, not all the three different types of states may appear in a generated FSM. For example, in the resulting FSM from the above formula depicted in Fig. 4, there is only one alert-state, indicated by the label $(-1,1)$, and two continue-states, indicated by the labels $(0,0)$ and $(1,1)$. The labels on the transitions are such that only positively interpreted propositions appear, whereas negative interpretations are implicit, for example, the label "inair_p" asserts the following valuation, $\{in\_air(p) = true, cross\_border(p) = false, landed(p) = false\}$. Note that the automatically generated FSMs are always complete in the sense that for all possible Boolean combinations of propositions there always exists a transition at each state. The transitions in the FSMs can often be simplified, for efficiency reasons. For instance, if a state has no outbound states but loops, we replace them with a single loop that is always enabled irrespective of the truth values of the propositions.

Instead of a direct implementation, FSMs are realized by reduction to description logic reasoning. This is done in a similar fashion as the encodings of LTL search control for planning in [Gab03]. For each state S in the FSA, we introduce a concept C_S into the semantic analysis knowledge base. For each of these concepts, we construct a DL formula by taking all the transitions (S_i, ψ_i, S) in the FSM and building a corresponding disjunction of the form $\bigsqcup_i C_{S_i} \sqcap \psi_i$. This formula essentially defines the extent (set of individuals in the corresponding state) of the concept C_S in the next time point. Every time a new ABox is loaded into the semantic analysis module, we use the current values of the concepts C_{S_i} and recompute the value of C_S by computing the answer to the query $\bigsqcup_i C_{S_i} \sqcap \psi_i$. This answer is then stored and loaded in the next time point so that the current values of the state concepts are always available for computing the values in the next time point. For example, for the formula above, we have three concepts C_1, C_2, C_3 meant to contain objects in the corresponding three states. The formulas for updating the concepts are, resp.: $(C_1 \sqcap \neg in\_air) \sqcup (C_2 \sqcap landed)$, $(C_2 \sqcap \neg landed \sqcap \neg cross\_border) \sqcup (C_1 \sqcap in\_air)$, $C_3 \sqcup (C_1 \sqcap cross\_border) \sqcup (C_2 \sqcap cross\_border)$. A comprehensive formal semantics of the alerts layer can also be given through a combination of DL and LTL as described in [BGL08].

2.4 Controlled Natural Language (CNL) Interface

A computer-understandable controlled natural language (CNL) is an engineered subset of a natural language designed to reduce ambiguity and vagueness that are inherent in full natural language. Our controlled natural language is based on an unification-based grammar similar to [FKK08, ST08] and relies on a neo-Davidsonian representation of events, and a small number of thematic roles that are used to link these events with other discourse entities (see [Par94] for an introduction).

The purpose of the CNL interface in the SAIL architecture is to allow humans who are not trained in formal logic to add eye-witness reports to the system and to query the DL knowledge base in CNL. This high-level interface abstracts away from primitive and defined concepts and from the formal notation used to encode these concepts in the knowledge base. Working with such an interface has the advantage that the user can employ the familiar terminology of the application domain to interact with the system, and we expect that this will reduce the cognitive load of the user considerably in a situation awareness context.

Implementation. The CNL processor of the SAIL system consists of a controlled lexicon and a bi-directional grammar. It translates declarative sentences and questions written in CNL into a formal representation in TPTP syntax [SS98], and it generates answers and alert messages in CNL. (We have chosen TPTP syntax for ease of interfacing reasoners.) The kernel of the CNL grammar is built around declarative sentences which have the following simple functional structure:

Subject + Predicate + [Object] + {Modifiers}

This functional pattern can be instantiated via a set of well-defined CNL constituents, for example:

Subject: (Su_24M) Predicate: (reaches) Object: (Bendeguz) Modifier: (within 6 minutes).

Starting from this simple sentence pattern, more complex sentences can be built in a systematic way using a number of constructors (for example coordinators and quantifiers). The CNL processor is able to resolve anaphoric references during the parsing process using the DL knowledge base. This results in a paraphrase that shows the user how the anaphoric expressions were interpreted. As an example consider the following eye-witness report:

SU_24M takes off from Becker-Bender at 09:00. The A50-1 takes off from Krupali at 09:30.
The fighter (SU_24M) flies towards Bendeguz. The AWACS (A50-1) flies towards Eaglevista.

Apart from a paraphrase, the CNL processor generates first a TPTP representation for this eye-witness report. This representation is then translated further into a suitable form to augment the existing information in the data aggregation layer.

The DL knowledge base can be queried in CNL. Questions usually have an inverted word order but the processing of questions can be interpreted as a variation of the processing of declarative sentences. Thus large parts of the same grammar can be used. Here is an example of a typical *wh*-question:

What aircraft of Redland is able to reach a city of Blueland?

The CNL processor translates this question into a TPTP formula and stores this formula as a template for generating answers:

```
input_formula(sail,conjunctive_query,((
 (? [A]: (named(A, s_redland) & (object(B, aircraft) &
                                property(B, associated_with, A)))) &
 (? [C]: ( (? [D]: (named(D, s_blueland) & (object(C, city) &
                                property(C, associated_with, D)))) &
 (? [E]: (property(E, has_agent, B) & (poss(E) & (event(E, reach) &
          (property(E, has_theme, C) & contemp(E, u))))))))
 => answer(B))).
```

The TPTP formula is then translated into a conjunctive nRQL query [HM03]:

```
(retrieve (?1) (and (?1 aircraft) (?1 s_redland associated_with)
  (?2 ?1 has_agent) (?2 reach) (?2 ?3 has_theme) (?3 city)
  (?3 s_blueland associated_with)))
```

This query is sent to the DL reasoner, RacerPro, for question answering. RacerPro returns the found instances. The CNL processor takes the stored TPTP formula and transforms it into a representation for a declarative sentence using these instances, and one or more complete sentences are generated as an answer.

3 Conclusions

We presented SAIL, a novel system architecture for situation awareness. It differs from other approaches by emphasizing the role of formal logics and automated reasoning systems. This supports a highly declarative approach to building situation awareness systems. To our knowledge, SAIL is the first approach of this kind, and we see the underlying approach of combining and extending *available* reasoners beyond their native capabilities as our main contribution. A core feature of our architecture is that it enable computation with data that changes over time, which is crucial for situation awareness but not natively supported by the reasoners. Additional components of the implementation are a GIS-system, a controlled natural language interface and Google-Earth visualization of trajectories and alerts.

We have implemented the SAIL system with the functionality described above. It copes with data streams from a realistic scenario in real time. The raw sensor data that feeds into the Data Aggregation layer arrives as a stream of time-stamped tuples `(T,Name,Ptfm,Alleg,Type,Lat,Long,Alt,VelX,VelY,VelZ,AccX,AccY,AccZ)` each describing a detected object's position with real geo-coordinates, physically real velocity and acceleration quantities, and, if known, its platform, allegiance (hostile, friendly or neutral), and type. As mentioned in Section 2, the stream delivers such information on about 30 objects approximately every 1/3sec. E-KRHyper's rule base consists of about 100 rules and ABoxes are generated at a rate of 2 ABoxes per "data-minute", where each ABox contains around 100-400 assertions. Each ABox is combined with the background knowledge assertions and the background knowledge axioms to form a single DL knowledge base. The background knowledge assertions are composed of 43 concept assertions and 28 role assertions. The background knowledge axioms are composed of 18 concept definitions, 14 inclusion axioms (GCIs) and 4 disjointness axioms. RacerPro is able to compute the result of each query in less than 2 real seconds.

As future work it would be interesting to consider applications in domains like air traffic control or disaster management.

Acknowledgements. We thank the reviewers and our DSTO project partner for valuable comments on earlier drafts of this paper.

References

[BCM+03] Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)

[BFP07] Baumgartner, P., Furbach, U., Pelzer, B.: Hyper tableaux with equality. In: Pfenning, F. (ed.) CADE 2007. LNCS, vol. 4603, pp. 492–507. Springer, Heidelberg (2007)

[BGL08] Baader, F., Ghilardi, S., Lutz, C.: LTL over description logic axioms. In: Brewka, G., Lang, J. (eds.) Procs. KR 2008. AAAI Press, Menlo Park (2008)

[BKS+06] Blasch, E., Kadar, I., Salerno, J., Kokar, M.M., Das, S., Powell, G.M., Corkill, D.D., Ruspini, E.H.: Issues and challenges in situation assessment (level 2 fusion). Journal of Advances in Information Fusion 1(2), 122–139 (2006)

[BLS06] Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 260–272. Springer, Heidelberg (2006)

[End95] Endsley, M.R.: Towards a theory of situation awareness in dynamic systems. Human Factors 37, 32 (1995)

[FKK08] Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)

[Gab03] Gabaldon, A.: Compiling control knowledge into preconditions for planning in the situation calculus. In: Procs. IJCAI 2003, Acapulco, Mexico (2003)

[GHGJ+07] Ghanea-Hercock, R.A., Gelenbe, E., Jennings, N.R., Smith, O., Allsopp, D.N., Healing, A., Duman, H., Sparks, S., Karunatillake, N.C., Vytelingum, P.: Hyperion—next-generation battlespace information services. The Computer Journal 50(6), 632–645 (2007)

[HM03] Haarslev, V., Möller, R.: Racer: A core inference engine for the semantic web. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870. Springer, Heidelberg (2003)

[MKL+05] Matheus, C.J., Kokar, M.M., Letkowski, J.J., Call, C., Baclawski, K., Hinman, M., Salerno, J., Boulware, D.: Lessons learned from developing SAWA: A situation awareness assistant. In: FUSION 2005: 7th International Conference on Information Fusion. IEEE, Los Alamitos (2005)

[NL05] Nowak, C., Lambert, D.: The semantic challenge for situation assessments. In: 8th International Conference on Information Fusion, July 2005. IEEE, Los Alamitos (2005)

[Par94] Parsons, T.: Events in the Semantics of English: A Study in Subatomic Semantics. MIT Press, Cambridge (1994)

[Pnu77] Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57. IEEE, Los Alamitos (1977)

[PW07] Pelzer, B., Wernhard, C.: System description: E- kRHyper. In: Pfenning, F. (ed.) CADE 2007. LNCS, vol. 4603, pp. 508–513. Springer, Heidelberg (2007)

[Ras94] Raschid, L.: A semantics for a class of stratified production system programs. Journal of Logic Programming 21(1), 31–57 (1994)

[SRS+07] Smart, P.R., Russell, A., Shadbolt, N.R., Shraefel, M.C., Aktivesa, L.A.C.: A technical demonstrator system for enhanced situation awareness. The Computer Journal 50(6), 704–716 (2007)

[SS98] Sutcliffe, G., Suttner, C.B.: The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning 21(2), 177–203 (1998)

[ST08] Schwitter, R., Tilbrook, M.: Meaningful Web Annotations for Humans and Machines using Controlled Natural Language. Expert Systems 25(3), 253–267 (2008)

On the Proof Theory of Regular Fixed Points

David Baelde

INRIA & LIX / École Polytechnique
david.baelde@ens-lyon.org

Abstract. We consider encoding finite automata as least fixed points in a proof-theoretical framework equipped with a general induction scheme, and study automata inclusion in that setting. We provide a coinductive characterization of inclusion that yields a natural bridge to proof-theory. This leads us to generalize these observations to *regular formulas*, obtaining new insights about inductive theorem proving and cyclic proofs in particular.

1 Introduction

The automated verification of systems that have only finitely many possible behaviors is obviously decidable, although possibly complex. More interestingly, many properties are still decidable in the much richer class where infinitely many behaviors can be described by a finite number of states. There are several reasons for considering these questions from a proof-theoretical angle. Obviously, proof-theory provides well-structured proof objects that can be considered as verification certificates; the fact that proofs can be composed by cut is of particular interest here. Taking the opposite point of view, complex but decidable problems can also be good examples to test and illuminate the design of rich logics.

In particular, we are interested in the treatment of finite-state behaviors in first-order logic extended with least fixed points. While the finite behavior case is trivially handled in the proof-theory of such logics, finite-state behaviors are not so well understood. Finite behaviors can be treated by only unfolding fixed points on both positive and negative positions. Applying an exhaustive proof-search strategy along these lines, the Bedwyr system [14,2] provides a purely syntactic approach to model-checking. Although simple, this strategy allows to treat complex problems like bisimulation for finite π-calculus, thanks to the seamless integration of generic quantification [8,13]. In order to deal with finite-state behaviors, a natural attempt is to detect cycles in proof-search and characterize those which reflect a sound reasoning. Following that general idea, tableau [5] and cyclic [10,12,4] proof systems have been explored under several angles. These systems are simple, especially natural from a semantic point of view, but not entirely satisfactory. Notably, they do not enjoy cut-elimination (except for the propositional framework of [10]) and, in the first-order, intuitionistic or linear cases, their cut-free proofs are not expressive enough for capturing finite-state behaviors.

In this paper, we first study the proof-theoretical treatment of finite automata inclusion, a central problem in model-checking, in a logic equipped with a general, explicit induction principle. We translate a finite automaton, or rather the acceptance of a word

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 93–107, 2009.

by that automaton, as an interleaved least fixed point predicate, and show that our simple framework offers a natural support for reasoning about such complex expressions. We then widen the scope of the discussion, investigating the completeness and the decidability of our logic for a more general class of finite-state behaviors. Such work can be rather technical, but we leverage the methodology and intuitions from the finite automata setting, eventually obtaining new insights about cyclic proofs.

The rest of the paper is organized as follows: we present in Section 2 the sequent calculus that we shall use, then study finite automata inclusion and its proof-theoretical support in Section 3, and finally generalize the methodology to *regular formulas* in Section 4. This work has been developed in Chapter 5 of the author's thesis [1]; we refer the reader to that document for complete proofs and more detailed discussions.

2 μMALL

We shall work with the logic μMALL [3,1], which is an extension of first-order linear logic without exponentials (MALL) with equality and least and greatest fixed points. The choice of linear logic might seem surprising, but we shall not use it as the logic of resource management but rather as a constrained framework underlying traditional reasoning. Indeed, we shall use the intuitionistic presentation of μMALL, which can easily and safely be read as usual intuitionistic logic. The point is that our observations will come out especially clearly in a linear framework. In this paper, we ignore greatest fixed points and consider only a first-order term language.

In the following, terms are denoted by s, t; vectors of terms are denoted by $\boldsymbol{s}, \boldsymbol{t}$; formulas (objects of type o) are denoted by P, Q, S; term variables are denoted by x, y. Finally, the syntactic variable B represents a formula abstracted over a predicate and n terms ($\lambda p \lambda x_1 \ldots \lambda x_n.\ Q p x_1 \ldots x_n$). We have the following formula constructors:

$$\begin{aligned} P ::= {} & P \otimes P \mid P \oplus P \mid P \multimap P \mid P \,\&\, P \mid \mathbf{1} \mid \mathbf{0} \mid \bot \mid \top \\ \mid {} & \exists_\gamma x.\ P \mid \forall_\gamma x.\ P \mid s \overset{\gamma}{=} t \mid \mu_{\gamma_1 \ldots \gamma_n}(\lambda p \lambda \boldsymbol{x}.\ P)\boldsymbol{t} \end{aligned}$$

The syntactic variable γ represents a term type, *e.g., nat*, *char*, *word*. The quantifiers have type $(\gamma \to o) \to o$ and the equality has type $\gamma \to \gamma \to o$. The least fixed point connective μ has type $(\tau \to \tau) \to \tau$ where τ is $\gamma_1 \to \cdots \to \gamma_n \to o$ for some arity $n \geq 0$. We shall almost always elide the references to γ, assuming that they can be determined from context when it is important to know their value. Formulas with top-level connective μ are called fixed point expressions and can be arbitrarily nested (which correspond to successive inductive definitions such as lists of natural numbers $\mu L.\ \mathbf{1} \oplus (\mu N.\ \mathbf{1} \oplus N) \otimes L)$ and interleaved (which corresponds to mutually inductive definitions such as arbitrarily branching trees $\mu T.\ \mathbf{1} \oplus (\mu L.\ \mathbf{1} \oplus T \otimes L))$. The first argument of a fixed point expression is called its *body*, and shall be denoted by B. In themselves, such second-order expressions are called *predicate operator* expressions or simply operators. We shall assume that *all bodies are monotonic*, *i.e.,* the bound predicate variables occur only positively in them.

We present the inference rules for μMALL in Figure 1. We omit the specification of the judgment $\Sigma \vdash t$ used in the right existential and left universal rules, expressing

Propositional fragment

$$\frac{\Sigma;\Gamma,P,P'\vdash Q}{\Sigma;\Gamma,P\otimes P'\vdash Q}\qquad\frac{\Sigma;\Gamma\vdash P\quad\Sigma;\Gamma'\vdash P'}{\Sigma;\Gamma,\Gamma'\vdash P\otimes P'}\qquad\frac{\Sigma;\Gamma\vdash P\quad\Sigma;\Gamma',P'\vdash Q}{\Sigma;\Gamma,\Gamma',P\multimap P'\vdash Q}\qquad\frac{\Sigma;\Gamma,P\vdash Q}{\Sigma;\Gamma\vdash P\multimap Q}$$

$$\frac{\Sigma;\Gamma,P\vdash Q\quad\Sigma;\Gamma,P'\vdash Q}{\Sigma;\Gamma,P\oplus P'\vdash Q}\qquad\frac{\Sigma;\Gamma\vdash P_i}{\Sigma;\Gamma\vdash P_0\oplus P_1}\qquad\frac{\Sigma;\Gamma,P_i\vdash Q}{\Sigma;\Gamma,P_0\,\&\,P_1\vdash Q}\qquad\frac{\Sigma;\Gamma\vdash P\quad\Sigma;\Gamma\vdash P'}{\Sigma;\Gamma\vdash P\,\&\,P'}$$

First-order structure

$$\frac{\Sigma,x;\Gamma,Px\vdash Q}{\Sigma;\Gamma,\exists x.Px\vdash Q}\qquad\frac{\Sigma\vdash t\quad\Sigma;\Gamma\vdash Pt}{\Sigma;\Gamma\vdash\exists x.Px}\qquad\frac{\Sigma\vdash t\quad\Sigma;\Gamma,Pt\vdash Q}{\Sigma;\Gamma,\forall x.Px\vdash Q}\qquad\frac{\Sigma,x;\Gamma\vdash Px}{\Sigma;\Gamma\vdash\forall x.Px}$$

$$\frac{\{\,\Sigma\theta;\Gamma\theta\vdash P\theta:\theta\in csu(u\doteq v)\,\}}{\Sigma;\Gamma,u=v\vdash P}\qquad\frac{}{\Sigma;\Gamma\vdash u=u}$$

Fixed points

$$\frac{\Sigma;\Gamma,S\,\boldsymbol{t}\vdash P\quad\boldsymbol{x};BS\,\boldsymbol{x}\vdash S\,\boldsymbol{x}}{\Sigma;\Gamma,\mu B\boldsymbol{t}\vdash P}\qquad\frac{\Sigma;\Gamma\vdash B(\mu B)\boldsymbol{t}}{\Sigma;\Gamma\vdash\mu B\boldsymbol{t}}\qquad\frac{}{\Sigma;\mu B\boldsymbol{t}\vdash\mu B\boldsymbol{t}}$$

Fig. 1. Intuitionistic presentation of first-order μMALL

that t is a well-formed term over the signature Σ. The initial identity rule is restricted to fixed points. In the left rule for μ, which is induction, S is called the invariant and is a closed formula of the same type as μB, of the form $\gamma_1\to\cdots\to\gamma_n\to o$. The treatment of equality dates back to [6,11]. In the left equality rule, *csu* stands for complete set of unifiers. Since we are only considering first-order terms in this paper, we have most general unifiers and hence at most one premise to this rule — there is none when the equality is absurd, *i.e.,* non-unifiable. Thanks to the monotonicity condition, the cut rule is admissible in μMALL [3], which implies the consistency of that system.

Example 1. We introduce the term type n for natural numbers, with two constants $0:n$ and $s:n\to n$. We define *nat* of type $n\to o$ and *half* of type $n\to n\to o$ as the fixed points μB_{nat} and μB_{half} where:

$$\begin{aligned}B_{nat}&\overset{def}{=}\lambda N\lambda x.\ x=0\oplus\exists y.\ x=s\,y\otimes N\,y\\B_{half}&\overset{def}{=}\lambda H\lambda x\lambda h.\ (x=0\otimes h=0)\oplus(x=s\,0\otimes h=0)\oplus\\&\qquad\qquad(\exists x'\exists h'.\ x=s\,(s\,x')\otimes h=s\,h'\otimes Hx'h')\end{aligned}$$

In the particular case of *nat*, the induction rule yields the usual induction principle:

$$\frac{\Gamma,S\ t\vdash P\qquad\dfrac{\vdash S\ 0\quad S\,y\vdash S(s\,y)}{(B_{nat}S)x\vdash S\,x}\ \oplus L,\exists L,\otimes L,=L}{\Gamma,nat\ t\vdash P}$$

Notice that reasoning takes place on the fixed point formula *nat*, not on the type n which could as well contain other irrelevant terms.

Although it is constrained by linearity, μMALL is a very expressive logic. It is easy to encode total runs of a Turing machine as a fixed point predicate, and hence provability in μMALL is undecidable in general.

3 Finite State Automata

Definition 1 (Finite state automaton, acceptance, language). *A non-deterministic finite state automaton $\mathcal{A}$ on the alphabet Σ is given by a tuple (Q, T, I, F) where Q is a set whose elements are called* states, *$T \in \wp(Q \times \Sigma \times Q)$ is a set of* transitions *and I and F are subsets of Q, respectively containing the* initial *and* final *states. A state q_0 is said to* accept *a word $\alpha_1 \ldots \alpha_n$ when there is a path $q_0 q_1 \ldots q_n$ where $q_n \in F$ and each $(q_{i-1}, \alpha_i, q_i) \in T$. The* language *$\mathcal{L}(q)$ associated to a state is the set of words that it accepts. That notion is extended to a collection of states by $\mathcal{L}(Q) := \cup_{q \in Q} \mathcal{L}(q)$ and to the automaton by $\mathcal{L}(\mathcal{A}) := \mathcal{L}(I)$.*

In the following we shall not specify on which alphabet each automaton works, supposing that they all work on the same implicit Σ, and α shall denote the letters of that alphabet. We also talk about transitions, final and initial states without making explicit the automaton that defines them, since it shall be recovered without ambiguity from the states[1]. We write $q \to^{\alpha} q'$ for $(q, \alpha, q') \in T$.

Definition 2 (α^{-1}). *For a language L and a set of states Q, we define α^{-1}:*

$$\alpha^{-1} L \overset{def}{=} \{w : \alpha w \in L\} \qquad \alpha^{-1} Q \overset{def}{=} \{q' : q \to^{\alpha} q' \textit{ for some } q \in Q\}$$

We finally come to the only non-standard definition

Definition 3 (Transitions between collections of states). *For a collection of states Q we write $Q \to^{\alpha} Q'$ when $Q' \subseteq \alpha^{-1} Q$.*

We now propose a coinductive characterization of inclusion. Its interest is to allow the transition from the (semantic) automata-theoretic inclusion to the (syntactic) inductive proof of implication in μMALL. As far as we know, that characterization is as novel as its purpose.

Definition 4 (Multi-simulation). *A multi-simulation between two automata (A, T, I, F) and (B, T', I', F') is a relation $\mathfrak{R} \subseteq A \times \wp(B)$ such that whenever $p \mathfrak{R} Q$:*

- *if p is final, then there must be a final state in Q;*
- *for any α and p' such that $p \to^{\alpha} p'$ there exists Q' such that $Q \to^{\alpha} Q'$ and $p' \mathfrak{R} Q'$.*

Proposition 1. *$\mathcal{L}(p) \subseteq \mathcal{L}(Q)$ if and only if $p \mathfrak{R} Q$ for some multi-simulation $\mathfrak{R}$.*

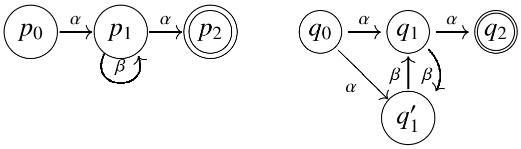
[1] States are mere identifiers, and automata are in fact considered modulo renaming. Hence, when several automata are considered we implicitly make the assumption that their sets of states are disjoints (a sort of Barendregt convention), which indeed makes it possible to recover the automaton from one of its states.

Proof. **If.** Let $\mathfrak{R}$ be a multi-simulation. We prove by induction[2] on the word w that whenever $p\mathfrak{R}Q$ and $w \in \mathcal{L}(p)$, then $w \in \mathcal{L}(Q)$. It is true for ϵ by definition, as there must be a final state in Q when p is final. If $w = \alpha w'$ is in $\mathcal{L}(p)$ then we have some p' such that $p \rightarrow^{\alpha} p'$, and by definition of multi-simulation there exists Q' such that $Q \rightarrow^{\alpha} Q'$ and $p'\mathfrak{R}Q'$. Since $w' \in \mathcal{L}(p')$ we obtain by induction hypothesis that $w' \in \mathcal{L}(Q')$ and hence $w \in \mathcal{L}(Q)$. **Only if.** We show that language inclusion is a multi-simulation, which follows immediately from the definition: if we have $\mathcal{L}(p) \subseteq \mathcal{L}(Q)$ and p is final then $\epsilon \in \mathcal{L}(Q)$, hence one of the states of Q must be final; if $p \rightarrow^{\alpha} p'$ then $\alpha\mathcal{L}(p') \subseteq \mathcal{L}(Q)$, that is $\mathcal{L}(p') \subseteq \alpha^{-1}\mathcal{L}(Q)$, and hence $Q' := \alpha^{-1}Q$ fits.

The inclusion is the greatest multi-simulation, the union of all multi-simulations. To obtain an illustrative proof that a given p is included in Q, it is more interesting to look at the least possible multi-simulation relating them.

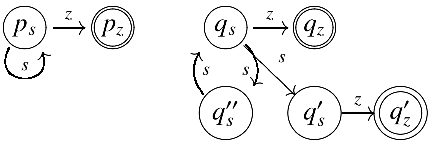
A multi-simulation establishing $\mathcal{L}(p_0) \subseteq \mathcal{L}(Q_0)$ can be obtained by iterating the following transformation on the relation $\{(p_0, Q_0)\}$: for each (p, Q) and each $p \rightarrow^{\alpha} p'$, add $(p', \alpha^{-1}Q)$ to the relation. When a fixed point is reached, check the condition on final states: if it holds the relation is a multi-simulation; if it does not there cannot be any multi-simulation relating p_0 and Q_0. This simple technique generally gives a smaller relation than inclusion, but it is still not the best.

Example 2. Consider the following two automata.



The state p_0 is included in q_0: if there is an even number of β transitions to make, go to q_1, otherwise go to q'_1. The inclusion is also "proved" by the multi-simulation $\mathfrak{R} = \{(p_0, \{q_0\}), (p_1, \{q_1, q'_1\}), (p_2, \{q_2\})\}$. One can sense here the richness of the multi-simulation technique, featuring the backwards aspect of proofs by induction.

Example 3. We finally show an example that hints at the upcoming generalization of this discussion. Informally, we shall prove the totality of *half* (*i.e.,* $\forall x.\ nat\ x \multimap \exists y.\ half\ x\ y$) by relying on an informal encoding of the behavior of $(\lambda x.\ nat\ x)$ and $(\lambda x \exists h.\ half\ x\ h)$ as automata:



The following multi-simulation establishes that $\mathcal{L}(p_s) \subseteq \mathcal{L}(q_s)$, *i.e., half* is total:

$$\mathfrak{R} = \{(p_s, \{q_s\}), (p_s, \{q'_s, q''_s\}), (p_z, \{q_z\}), (p_z, \{q'_z\})\}$$

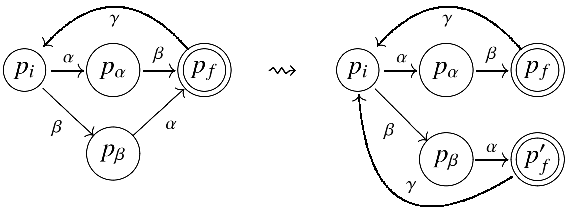
[2] The notion of multi-simulation extends naturally to labeled transition systems. In that case, both finite and infinite trace inclusions would be multi-simulations. However, multi-simulation only implies the inclusion of *finite* traces — which is shown by induction on the length of the trace.

We have exhibited a simple structure underlying inclusion of non-deterministic finite automata, which expresses non-trivial reasoning. We are now going to move to the (linear) logical world and exploit it.

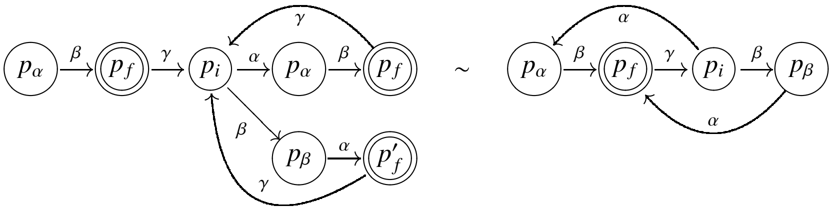
3.1 Encoding Finite Automata in μMALL

We shall represent an automaton, or rather its acceptance predicate, in the logic μMALL. A first possibility would be to encode one automaton as one fixed point taking both the state and the word as arguments: the resulting fixed point would be large but simply structured. We are interested in a more *structural* approach, that erases the names of states and translates each state of an automaton as one fixed point expression in which other states are also expressed, in a complex interleaved way. Our interest is to test the logic on this large class of interleaved fixed points, and then generalize our observations to a wider class of fixed points.

The drawback of encoding to interleaved fixed points is that it forces a sequential introduction of mutually inductive predicates, which forbids sharing. Graphically, that sequentialization requires writing a graph as a tree (the syntax tree) plus looping edges (predicate variables referring to fixed points). For example, the following transformation will essentially be applied when encoding p_i as a μ-formula:



Taking the α transition from p_i involves an unfolding of the μ formula, corresponding to the left automaton below. It is different from the automaton corresponding to a direct translation of state p_α in the initial automaton, shown on the right below. But, the two automata are bisimilar.



We now describe formally the translation. Note that it could be generalized to encode mutually (co)inductive definitions into fixed points, and most observations made here would still be relevant [1].

Definition 5 (Translation of automata into fixed points). *Let $\mathcal{A}$ be a finite automaton. We translate each of its states q into a fixed point predicate expression $[q]^{\emptyset}$ as follows:*

$$[q_i]^\Gamma \equiv \begin{cases} q_i & \textit{if } q_i \in \Gamma \\ \mu(\lambda q_i \lambda w.\ \{\ w = \epsilon\ :\ q_i \textit{ is final}\ \} \bigoplus \{\ \exists w'.\ w = \alpha w' \otimes [q_j]^{\Gamma, q_i} w'\ :\ q_i \rightarrow^\alpha q_j\ \}) & \end{cases}$$

This encoding is structural: it does not rely on the names of the defined atoms but only reflects their structure. As we have shown in the previous examples, bisimilar states might be identified, and structurally identical states might only have "bisimilar" encodings.

Proposition 2 (Adequacy). *Let $\mathcal{A}$ be a finite automaton. There is a bijection between accepting paths starting at one of its state q and cut-free μMALL derivations of $\vdash [q]w$, where w is the word induced by the path.*

Our encoding does not only provide adequate representations, but also powerful ways of reasoning about automata. The following deduction rule gives a very natural synthetic reading of an automaton's encoding.

Proposition 3. *Let A be a finite automaton, and p one of its states. The following rule is sound and invertible, where the states p', p'' are taken among those reachable from p:*

$$\frac{\{\vdash S_{p'}\epsilon : p' \textit{ final}\} \quad \{S_{p''}x \vdash S_{p'}(\alpha x) : p' \rightarrow^\alpha p''\} \quad \Gamma, S_0 t \vdash Q}{\Gamma, [p]t \vdash Q}$$

The rule is derived by repeatedly applying (asynchronous) rules on the state's encoding. The final clauses occur naturally. The transition clauses occur in two different ways, depending whether the arc between two states is in the covering tree or is a looping arc. Like the induction rule, this rule leaves the difficult choice of finding a correct invariant for each state, and it is not invertible for all choices. A typical example is to use the unfolded fixed points as invariants, which yields the invertible rule of case analysis.

Theorem 1 (Soundness and completeness). *Let $\mathcal{A}$ and $\mathcal{B}$ be two automata, let p_0 be a state of $\mathcal{A}$ and Q_0 a collection of states of $\mathcal{B}$. Then $\mathcal{L}(p_0) \subseteq \mathcal{L}(Q_0)$ if and only if $\forall w.\ [p_0]w \multimap \oplus_{q \in Q_0} [q]w$ is provable in μMALL.*

Proof. The easy direction here is to conclude the inclusion from the provability of the linear implication. It immediately follows from the adequacy of the encoding and cut-elimination. For the other direction we use our characterization of inclusion. Since $\mathcal{L}(p_0) \subseteq \mathcal{L}(Q_0)$ there exists a multi-simulation $\mathfrak{R}$ that relates them. We prove $[p_0]w \vdash \oplus_{q \in Q_0} [q]w$ by using the simultaneous induction rule shown above, with the invariants S_p given as follows by the multi-simulation:

$$S_p := \lambda x.\ \&_{p \mathfrak{R} Q} \oplus_{q \in Q} [q]x$$

We have to build a proof of $\vdash S_p\epsilon$ for each terminal state p: we enumerate all $p\mathfrak{R}Q$ by introducing the &, then since $\mathfrak{R}$ is a multi-simulation there must be a final state $q_f \in Q$, we select this disjunct and finally derive $\vdash [q_f]\epsilon$ by selecting the final clause in it.

We also have to build a derivation of $S_{p'}x \vdash S_p(\alpha x)$ for each $p \rightarrow^\alpha p'$. We introduce again the & on the right, enumerating all $p\mathfrak{R}Q$, then by definition of the multi-simulation there is a Q' such that $Q \rightarrow^\alpha Q'$ and $p'\mathfrak{R}Q'$, so we choose the corresponding

&-conjunct on the left hand-side. We are left with $\oplus_{q' \in Q'} \; [q']x \vdash \oplus_{q \in Q} \; [q](\alpha x)$ which corresponds exactly to $Q \rightarrow^{\alpha} Q'$: for each q' there is a q such that $q \rightarrow^{\alpha} q'$. We translate that by enumerating all q' (introducing the left $\oplus$) and choosing the appropriate q in each branch (introducing the right $\oplus$) and finally selecting the right clause in $[q]$ to establish $[q']x \vdash [q](\alpha x)$.

Our representation of automata as fixed points is very satisfying: the derived induction principle allows rich reasoning about acceptances, naturally reflecting the behavior of an automaton. It fits perfectly with the notion of multi-simulation. This would be less precise in the richer framework of intuitionistic logic: the encoding and adequacy result can be adapted straightforwardly, and the linear derivation built in the above proof can be mimicked by a similar intuitionistic derivation to obtain the same theorem, but new possible behaviors involving an additive treatment of the conjunction would be irrelevant.

4 Regular Formulas

We obtained a completeness result for finite automata, based on the construction of complex invariants from multi-simulations, which can be discovered automatically. It is tempting to extend such a good property. In this section, we consider a notion of *regular formulas* that is considerably richer than encodings of finite automata, in an attempt to capture simple but useful properties such as totality of relations. Like finite automata, regular formulas are finite systems of interdependent superficial constraints. The main differences are that regular formulas deal with terms rather than words and have an arbitrary arity. We shall see that the latter extension makes regular formulas much more complex than finite automata.

While only the first letter of a word is checked when taking a transition in an automata, terms in regular formulas are matched against arbitrary patterns. In particular, patterns can be trivial, which corresponds to ϵ-transitions.

Definition 6 (Patterns). *A* pattern *C of type $\gamma_1, \ldots, \gamma_n \rightarrow \gamma'_1, \ldots, \gamma'_m$ is a vector of m closed terms*[3] *$p_i : \gamma_1, \ldots, \gamma_n \rightarrow \gamma'_i$, such that each of the n variables occurs at most once in all $(p_i)_i$. The (p_i) are called* elementary patterns *of C. A pattern is said to be* non-erasing *when each of its input variables occurs in one of its elementary patterns.*

We write $C\boldsymbol{t}$ for denoting the vector resulting from the application of $\boldsymbol{t}$ to each elementary pattern of C. For two vectors of terms of equal length n, $\boldsymbol{t} = \boldsymbol{t'}$ denotes the formula $t_1 = t'_1 \otimes \ldots \otimes t_n = t'_n$. The patterns C and C' are said to be compatible *when the unification problem $C\boldsymbol{x} = C'\boldsymbol{y}$ has a solution.*

A trivial pattern *is a pattern which has no rigid structure,* i.e., *whose elementary patterns are projections. Trivial patterns are denoted by ϵ. A particular case of trivial pattern is the identity: we denote by $\mathcal{I}_n$ the pattern $(\lambda \boldsymbol{x}.x_1, \ldots, \lambda \boldsymbol{x}.x_n)$.*

[3] The p_i are not first-order terms but they are only a technical device for presenting patterns. The point is that they shall always be applied when occurring in formulas, hence yielding terms of ground type γ'.

Definition 7 (Pattern compositions). *Let C and C' be patterns of arbitrary type. Let $(p_i)_{i \leq m}$ be the elementary patterns of C and $(p'_j)_{j \leq m'}$ those of C'. We define (C, C') to be $(\lambda \boldsymbol{xy}.p_1\boldsymbol{x}, \ldots, \lambda \boldsymbol{xy}.p_m\boldsymbol{x}, \lambda \boldsymbol{xy}.p'_1\boldsymbol{y}, \ldots, \lambda \boldsymbol{xy}.p'_{m'}\boldsymbol{y})$, which is still a pattern.*

Assuming that C has type $\boldsymbol{\gamma} \to \boldsymbol{\gamma'}$, and C' has type $\boldsymbol{\gamma'} \to \boldsymbol{\gamma''}$, we define $C'C$ to be the pattern $(\lambda \boldsymbol{x}.p'_1(C\boldsymbol{x}), \ldots, \lambda \boldsymbol{x}.p'_{m'}(C\boldsymbol{x}))$.

Definition 8 (Regular formula). *We define the class of formulas $\mathcal{R}^{\Gamma}_{I/O}$, parametrized by Γ (a set of predicate variables), I (a set of* input *term variables) and O (a set of* output *variables). The* regular formulas *on a signature Σ are given by $\mathcal{R}^{\emptyset}_{\emptyset/\Sigma}$.*

$$
\begin{aligned}
\mathcal{R}^{\Gamma}_{I/O} ::= {} & \mathcal{R}^{\Gamma}_{I/O} \oplus \mathcal{R}^{\Gamma}_{I/O} \mid \exists y.\, \mathcal{R}^{\Gamma}_{I,y/O} \mid \mathcal{P}^{\Gamma}_{I \cup O} \\
\mid {} & O = C \ \ \textit{when } I = \emptyset \\
\mid {} & O' = CI \otimes \mathcal{P}^{\Gamma}_{I \cup O''} \ \ \textit{when } O' \textit{ and } O'' \textit{ form a partition of } O \\
\mathcal{P}^{\Gamma}_{I} ::= {} & p\boldsymbol{x} \mid \mu(\lambda p.\lambda \boldsymbol{x}.\, \mathcal{R}^{\Gamma,p}_{\emptyset/\boldsymbol{x}})\boldsymbol{x} \ \ \textit{where } \boldsymbol{x} \textit{ is } I \textit{ in an arbitrary order}
\end{aligned}
$$

We say that a predicate P is regular when $P\boldsymbol{x}$ is regular over the signature $\boldsymbol{x}$.

The syntactic definition of regular formulas is quite restrictive but suffices to capture interesting examples. In particular, encodings of finite automata are regular formulas. In the last clause of $\mathcal{R}$, notice that the splitting of O allows that some unconstrained output variables are passed to the recursive occurrence $\mathcal{P}$. This allows direct encodings of ϵ-transitions, without resorting to an artificial clause of the form $\lambda w.\ \exists w'.\ w = w' \otimes p'\ w'$ for copying the input variable to the output.

Notice that the fixed point subformulas of a regular formula do not have free term variables. Hence, regular formulas can be seen as encodings of definitions [6,11,7,9] allowing mutual inductive definitions.

Example 4. Both $(\lambda x.\ nat\ x)$ and $(\lambda x.\ \exists h.\ half\ x\ h)$ are regular predicates. The usual specification of addition would also be regular, but not that of multiplication. It is also not possible to encode automata with (unbounded) state, as it would require to pass constructed terms to recursive occurrences of fixed points.

We now exhibit a fundamental property of regular formulas, which shall allow us to abstract away from their tedious syntactic definition.

Proposition 4 (Fundamental property). *Let P be a regular predicate. There is a finite collection of (regular) predicates (P_i), called* states *of P, such that P_0 is P and:*

- *Each $P_i\boldsymbol{x}$ is provably equivalent to an additive disjunction of formulas of the form $\exists \boldsymbol{y}.\ \boldsymbol{x} = C\boldsymbol{y}$ or $\exists \boldsymbol{y}.\ \boldsymbol{x} = C\boldsymbol{y} \otimes P_j\boldsymbol{y}$:*

$$\forall \boldsymbol{x}.\ (P_i\boldsymbol{x} \multimapboth (\exists \boldsymbol{y}.\ \boldsymbol{x} = C\boldsymbol{y}) \oplus (\exists \boldsymbol{y}.\ \boldsymbol{x} = C'\boldsymbol{y} \otimes P_j\boldsymbol{y}) \oplus \ldots)$$

  When the first form occurs in the disjunction we say that P_i is C-final; when the second one occurs we write $P_i \to^{C'} P_j$.
- *The following rule is admissible:*

$$\frac{\{ \vdash S_iC \ : P_i\ C\text{-final} \} \quad \{ \boldsymbol{x}; S_j\boldsymbol{x} \vdash S_i(C\boldsymbol{x}) \ : P_i \to^{C} P_j \} \quad \Gamma, S_0\boldsymbol{t} \vdash Q}{\Gamma, P\boldsymbol{t} \vdash Q}$$

Proof. The finite decomposition in states comes from the fact that only finitely many formulas can occur when exploring a regular formula by unfolding its fixed points, except for the term parameters which are however decreasing in size. This is because the unfoldings are only done superficially, as needed. A corollary of this is the decidability of the provability of $\vdash P\boldsymbol{t}$, and more generally of $\vdash \exists \boldsymbol{x}.\ P(C\boldsymbol{x})$.

For proving a regular formula P by induction on another regular formula Q, we need to adapt the states of P so that their behavior is finitely defined for the transitions of Q, which might be finer than those of P.

Definition 9 **(Q-states).** *Let P and Q be regular predicates of the same type. We say that P admits Q-*states *if there is a finite number of predicates (P'_i) such that:*

- *P is equivalent to P'_0;*
- *for each transition C of Q, each P'_i of compatible type, $P'_i(C\boldsymbol{x})$ is provably equivalent to an additive disjunction of $P'_j\boldsymbol{x}$.*

Theorem 2 **(Internal completeness).** *Let P and Q be two regular predicates of same type such that P admits Q-states, then $\{\ \boldsymbol{t} : \vdash Q\boldsymbol{t}\ \} \subseteq \{\ \boldsymbol{t} : \vdash P\boldsymbol{t}\ \}$ if and only if $\boldsymbol{x}; Q\boldsymbol{x} \vdash P\boldsymbol{x}$.*

Proof. If we have a derivation of the implication we obtain the inclusion by cut-elimination. For the other direction, we use a technique similar to the first part of the proof of Proposition 1.

When P' and Q' are predicates of the same type, we simply write $Q' \subseteq P'$ for $\{\ \boldsymbol{t} : \vdash Q'\boldsymbol{t}\ \} \subseteq \{\ \boldsymbol{t} : \vdash P'\boldsymbol{t}\ \}$. We shall build a derivation that uses the derived induction rule on Q (Proposition 4). Consider the Q-states of P, called $(P'_i)_{i\leq n}$. For each state Q_i, we form the conjunction of all unions of Q-states of P that contain Q_i:

$$S_i := \&\{\ \oplus_k P'_{i_k}\ :\ Q_i \subseteq \oplus_k P'_{i_k}\ \}$$

We check that the (S_i) are valid invariants for Q:

- For each C-final Q_i, we have by definition of S_i an acceptance of C in each conjunct, which allows us to prove $\vdash S_i C$.
- For each transition $Q_i \rightarrow^C Q_j$, we need to derive $S_j\boldsymbol{x} \vdash S_i(C\boldsymbol{x})$. Our derivation starts by a & rule which enumerates all conjuncts S of S_i. Each S contains Q_i by definition of S_i, and by definition of the Q-states there is another disjunction of Q-state S' such that $S'\boldsymbol{x} \multimapboth S(C\boldsymbol{x})$.
  We observe that Q_j is contained in S': If Q_j accepts $\boldsymbol{t}$ then Q_i accepts $C\boldsymbol{t}$, and so does S; By cutting this against the above equivalence we obtain that S' accepts $\boldsymbol{t}$. So we have S' in S_j, and we select this conjunct on the left hand-side. We now have to derive $S'\boldsymbol{x} \vdash S(C\boldsymbol{x})$ which is simply the other direction of the above equivalence.

As for the corresponding proof about finite automata, this proof yields a (naive) decision procedure: there is only a finite number of invariants to try, and it is decidable to check the final premises, as well as the transition premises since their form is very limited. As for multi-simulation on automata, the full invariant considered in our proof of internal completeness is often unnecessarily large, but more economic techniques apply equally well on Q-states.

Unfortunately, it is not always possible to obtain Q-states. We propose a partial procedure for computing them, then discuss in which cases it might fail.

Algorithm 11 (Partial procedure for computing Q-states) . *Let P and Q be regular predicates, and (P_i) be the states of P. We denote by P_i^* a reordering of the arguments of P_i,* i.e., *P_i^* is $(\lambda(x_k)_k.\ P_i(x_{\sigma(k)})_k)$ for some permutation σ. Note that the characterization of states of Proposition 4, can be adapted to be of the form $\forall \mathbf{x}.\ (P_i\mathbf{x} \multimapboth (\mathbf{x} = C) \oplus (\exists \mathbf{y}.\ \mathbf{x} = C'\mathbf{y} \otimes \exists \mathbf{y}'.\ P_j^*\mathbf{y}\mathbf{y}') \oplus \ldots$ where C' is non-erasing. We shall use that form in the proof below.*

The algorithm generates Q-states of the form $(\lambda \mathbf{x}.\ \mathbf{x} = C)$ or $(\lambda \mathbf{x}.\ \exists \mathbf{y}.\ \mathbf{x} = C\mathbf{y} \otimes \exists \mathbf{z}.\ P_j^*\mathbf{y}\mathbf{z})$ for some state P_j and a non-erasing pattern C. Strictly speaking, we need to generalize slightly over that format in order to handle erasing transitions of Q: we allow extra vacuous abstractions at any position, but limit the total arity to not exceed that of the transitions of C. This is shallow, and can be ignored in the following by considering the non-erasing restriction of a transition of Q, and adjusting the corresponding decomposition afterwards.

We build a set of Q-states as the fixed point[4] of the following transformation, starting with the singleton $\lambda \mathbf{x}.\ \exists \mathbf{y}.\ \mathbf{x} = \mathbf{y} \otimes P_0\mathbf{y}$. The transformation consists in computing a decomposition of the right form for each P'_i of our tentative set of Q-states and each transition C_Q of Q, and adding the components of the decomposition to our collection:

- *If P'_i is of the form $(\lambda \mathbf{x}.\ \mathbf{x} = C)$ then $P'_i(C_Q\mathbf{x})$ is provably equivalent to some $\mathbf{x} = C'$ if C_Q and C are compatible, which degenerates into $\mathbf{1}$ when $C_Q = C$ and $\mathbf{x}$ is empty; and it is equivalent to $\mathbf{0}$ if the patterns are incompatible. In both cases we have a valid decomposition, empty in the second case.*
- *Otherwise, our Q-state P' is of the form $(\lambda \mathbf{x}.\ \exists \mathbf{y}.\ \mathbf{x} = C\mathbf{y} \otimes \exists \mathbf{z}.\ P_i^*\mathbf{y}\mathbf{z})$.*
    - *If C_Q and C are incompatible, $P'(C_Q\mathbf{x})$ is simply equivalent to $\mathbf{0}$.*
    - *If C_Q has no rigid structure, it is enough to observe that $P'(C_Q\mathbf{x}) \multimapboth P'^*(\mathbf{x})$.*
    - *If C has no rigid structure, $P'(C_Q\mathbf{x})$ is equivalent to $\exists \mathbf{z}.\ P_i^*(C_Q\mathbf{x})\mathbf{z}$. The predicate P_i is equivalent to its characterization as a state,* i.e., *the sum of all its transitions and final patterns: $P_i\mathbf{x}' \multimapboth (\oplus_j\ F_j\mathbf{x}') \otimes (\oplus_k\ T_k\mathbf{x}')$. We decompose recursively[5] each $F_j^*(C_Q\mathbf{x})\mathbf{z}$ and $T_k^*(C_Q\mathbf{x})\mathbf{z}$ as a sum of Q-states, and manipulate the results to obtain a decomposition for $P'(C_Q\mathbf{x})$.*

      Our $P'(C_Q\mathbf{x})$ is equivalent to $\exists \mathbf{z}.\ \oplus_k\ (P''_k\mathbf{x}\mathbf{z})$, that is $\oplus_k\ \exists \mathbf{z}.\ (P''_k\mathbf{x}\mathbf{z})$. It remains to adapt the disjuncts into well-formed Q-states. We only show how to treat the case of a transition clause P''_k, the treatment of a final clause being a particular case. We start with:

      $$\exists \mathbf{z}.\ \exists \mathbf{y}'.\ (\mathbf{x}, \mathbf{z}) = C'\mathbf{y}' \otimes \exists \mathbf{z}'.\ P_j^*\mathbf{y}'\mathbf{z}'$$

      Splitting C' into (C'_1, C'_2) and $\mathbf{y}'$ into $(\mathbf{y}'_1, \mathbf{y}'_2)$ accordingly, we obtain:

      $$\exists \mathbf{y}'_1.\ \mathbf{x} = C'_1\mathbf{y}'_1 \otimes \exists \mathbf{z} \exists \mathbf{y}'_2 \exists \mathbf{z}'.\ \mathbf{z} = C'_2\mathbf{y}'_2 \otimes P_j^*\mathbf{y}'_1\mathbf{y}'_2\mathbf{z}'$$

[4] The iteration might diverge, if there is no finite fixed point.

[5] This recursive decomposition can loop if there is a cycle of trivial transitions in the states of P.

Finally, we can remove the useless information about $\boldsymbol{z}$*, without loosing the equivalence:*

$$\exists \boldsymbol{y}'_1.\ \boldsymbol{x} = C'_1 \boldsymbol{y}'_1 \otimes \exists \boldsymbol{y}'_2 \exists \boldsymbol{z}'.\ P^*_j \boldsymbol{y}'_1 \boldsymbol{y}'_2 \boldsymbol{z}'$$

- *When* C_Q *and* C *both have some rigid structure, then* $C_Q\boldsymbol{x} = C\boldsymbol{y}$ *can be decomposed into* $\boldsymbol{x}_1 = C'\boldsymbol{y}_1 \otimes \boldsymbol{y}_2 = C'_Q\boldsymbol{x}_2$*, where* $\boldsymbol{x}_1, \boldsymbol{x}_2$ *(resp.* $\boldsymbol{y}_1, \boldsymbol{y}_2$*) is a partition of* $\boldsymbol{x}$ *(resp.* $\boldsymbol{y}$*). This decomposition is obtained by destructing the common rigid structure of* C *and* C_Q*, aggregating in* C' *(resp.* C'_Q*) the residual constraints corresponding to branches where* C_Q *(resp.* C*) becomes flexible first.*
  So we have an equivalence between $P'(C_Q\boldsymbol{x})$ *and:*

$$\exists \boldsymbol{y}_1\boldsymbol{y}_2.\ \boldsymbol{x}_1 = C'\boldsymbol{y}_1 \otimes \boldsymbol{y}_2 = C'_Q\boldsymbol{x}_2 \otimes \exists \boldsymbol{z}.\ P^*_i \boldsymbol{y}_1\boldsymbol{y}_2\boldsymbol{z}$$

Or simply:

$$\exists \boldsymbol{y}_1.\ \boldsymbol{x}_1 = C'\boldsymbol{y}_1 \otimes \exists \boldsymbol{z}.\ P^*_i \boldsymbol{y}_1(C'_Q\boldsymbol{x}_2)\boldsymbol{z}$$

We recursively[6] *compute the decomposition of* P^*_i *for the pattern* $(\mathcal{I}_{|\boldsymbol{y}_1|}, C'_Q, \mathcal{I}_{|\boldsymbol{z}|})$*. As before, we shall obtain a decomposition of* P' *from that of* P^*_i*. We detail the case of transition clauses, for which we obtain a disjunct of the following form:*

$$\begin{aligned}&\exists \boldsymbol{y}_1.\ \boldsymbol{x}_1 = C'\boldsymbol{y}_1 \otimes \\ &\qquad\qquad \exists \boldsymbol{z}.\ \exists \boldsymbol{y}'_{\mathbf{1}} \exists \boldsymbol{y}'_{\mathbf{2}} \exists \boldsymbol{y}'_{z}.\ (\boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{z}) = (C_1\boldsymbol{y}'_{\mathbf{1}}, C_2\boldsymbol{y}'_{\mathbf{2}}, C_z\boldsymbol{y}'_{z}) \otimes \exists \boldsymbol{z}'.\ P^*_j \boldsymbol{y}'_{\mathbf{1}}\boldsymbol{y}'_{\mathbf{2}}\boldsymbol{y}'_{z}\boldsymbol{z}'\end{aligned}$$

We combine patterns:

$$\exists \boldsymbol{y}'_{\mathbf{1}}\boldsymbol{y}'_{\mathbf{2}}.\ (\boldsymbol{x}_1, \boldsymbol{x}_2) = (C'(C_1\boldsymbol{y}'_{\mathbf{1}}), C_2\boldsymbol{y}'_{\mathbf{2}}) \otimes \exists \boldsymbol{z}.\ \exists \boldsymbol{y}'_{z}.\ \boldsymbol{z} = C'_z\boldsymbol{y}'_z \otimes \exists \boldsymbol{z}'.\ P^*_j \boldsymbol{y}'_{\mathbf{1}}\boldsymbol{y}'_{\mathbf{2}}\boldsymbol{y}'_{z}\boldsymbol{z}'$$

And finally remove the constraint on hidden variables $\boldsymbol{z}$*, to obtain a decomposition of the right form:*

$$\exists \boldsymbol{y}'_{\mathbf{1}}\boldsymbol{y}'_{\mathbf{2}}.\ (\boldsymbol{x}_1, \boldsymbol{x}_2) = (C'(C_1\boldsymbol{y}'_{\mathbf{1}}), C_2\boldsymbol{y}'_{\mathbf{2}}) \otimes \exists \boldsymbol{y}'_{z}.\ \exists \boldsymbol{z}'.\ P^*_j \boldsymbol{y}'_{\mathbf{1}}\boldsymbol{y}'_{\mathbf{2}}\boldsymbol{y}'_{z}\boldsymbol{z}'$$

As is visible in the definition, several problems can cause the divergence of our algorithm. We propose some constraints under which it terminates, but also show why it is interesting in a more general setting.

Proposition 5. *Let* P *be a regular predicate such that its states have an arity of at most one, and there is no cycle of* ϵ*-transitions in it. Then it admits* Q*-states for any regular* Q*. Hence the derivability of* $\forall x.\ Qx \multimap Px$ *is decidable, and holds whenever* $Q \subseteq P$*.*

Proof. Algorithm 11 clearly returns valid Q-states when it terminates, and it is easy to show that it does terminate under the assumptions on P. Hence, Theorem 2 applies.

A regular formula constrained as in the previous proposition is not much more than a finite automaton: we have essentially shown that Theorem 1 is a particular case of the results on regular formulas. A noticeable difference is the ϵ-acyclicity condition: there is no difficulty in extending directly the work on finite automata to handle ϵ-transitions, but handling ϵ-cycles in Algorithm 11 involves some extra technicalities [1].

[6] This can create a loop if C'_Q does not decrease, *i.e.*, if C only has rigid structure on components where C_Q does not.

Example 5. In Proposition 5, the condition on the arity of P is essential. We show a simple binary example where our procedure diverges. Consider the regular predicates $P := \mu P \lambda x \lambda y.\ \exists x' \exists y'.\ x = s^2 x' \otimes y = s y' \otimes P x' y'$, and a predicate Q with a transition $(\lambda x'.\ s x', \lambda y.\ s y')$. We compute the Q-states of P using Algorithm 11. We start with P itself, and there is only one transition to consider: $(\lambda xy.sx, \lambda xy.sy)$. This is a rigid-rigid case, the decomposition of $p\ (sx)\ (sy)$ yields $\exists x' \exists y'.\ x = s x' \otimes y = y' \otimes p\ x'\ y'$. This new Q-state has to be decomposed for the same transition, and we obtain $\exists x' \exists y'.\ x = s^2 x' \otimes y = y' \otimes p\ x'\ y'$. The same pattern keeps applying, with information accumulating on x, and new Q-states keep being generated.

Example 6. In Example 3, we gave an informal proof of the totality of *half* by seeing *nat* and *half* as finite automata. We can now avoid that step and obtain directly a derivation. The states of $H \equiv \lambda x. \exists h.\ half\ x\ h$ are $H_0 \equiv \lambda x. \exists h.\ half\ x\ h$ and $H_1 \equiv \lambda x \lambda h.\ half\ x\ h$. Its *nat*-states can be obtained by our procedure:

$$\begin{array}{ll} H'_0 \equiv \lambda x.\ \exists h.\ half\ x\ h & \qquad H'_2 \equiv \lambda x.\ \exists p.\ x = sp \otimes \exists h.\ half\ p\ h \\ H'_1 \equiv \lambda x.\ x = 0 & \qquad H'_3 \equiv \lambda x.\ \mathbf{1} \end{array}$$

Starting from H'_0, and taking the successor transition of *nat*, we obtain H'_1 and H'_2 corresponding to the two transitions of H_1 that are compatible with the successor. Finally, H'_3 is obtained for the decomposition of all others against the zero transition. Notice that it is crucial that our algorithm eliminates the information learned about h as some constraints are applied on x, otherwise we could not obtain a finite number of *nat*-states.

Applying the proof of completeness, we essentially obtain the following invariant of *nat*, from which one can simply derive the totality of *half*:

$$S := \lambda x.\ (\exists h.\ half\ x\ h)\ \&\ (x = 0 \oplus \exists y.\ x = s\ y \otimes \exists h.\ half\ y\ h)$$

4.1 Relationship to Cyclic Proofs

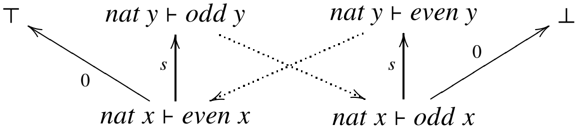
As said above, cyclic proofs are appealing from an automated reasoning perspective, as they avoid the invention of invariants, but they are very weak. It is our hope that the work presented in this paper eventually leads to useful theorem proving techniques that would keep the practical aspect of cyclic proofs but be much more powerful, in particular be complete for inclusions of automata, and still meaningful for regular formulas.

On the particular problem of inclusion, cyclic proofs seem related to simulations, associating one state with another. This is not enough for obtaining completeness. In contrast to that, the proofs obtained from our completeness theorems use invariants that express the less restrictive notion of multi-simulation, where one state can be related to several. This offers an interesting trade-off between expressiveness and the subformula property, since the invariants under consideration are still built from subformulas of the goal (its states).

It is interesting to present our proofs in an extended cyclic style, which takes into account failures and alternatives, as well as loops between alternatives. Consider the following example, for which there is no cut-free cyclic proof:

$$\dfrac{\boxed{\dfrac{\dfrac{}{\vdash \mathit{even}\ 0} \qquad \dfrac{\dfrac{\infty}{\mathit{nat}\ y \vdash \mathit{odd}\ y}}{\mathit{nat}\ y \vdash \mathit{even}\ (sy)}}{\mathit{nat}\ x \vdash \mathit{even}\ x}} \oplus \boxed{\dfrac{\dfrac{\bot}{\vdash \mathit{odd}\ 0} \qquad \dfrac{\dfrac{\infty}{\mathit{nat}\ y \vdash \mathit{even}\ y}}{\mathit{nat}\ y \vdash \mathit{odd}\ (sy)}}{\mathit{nat}\ x \vdash \mathit{odd}\ x}}}{\mathit{nat}\ x \vdash \mathit{even}\ x \oplus \mathit{odd}\ x}$$

Establishing the correctness of such objects is not trivial, but we probably have most of the tools at hand. It is indeed certainly related to the inclusion of *nat* in the underlying automata:


⊤
$\mathit{nat}\ y \vdash \mathit{odd}\ y$
$\mathit{nat}\ y \vdash \mathit{even}\ y$
⊥
0
s
s
0
$\mathit{nat}\ x \vdash \mathit{even}\ x$
$\mathit{nat}\ x \vdash \mathit{odd}\ x$

Our work on regular formulas shows that there are two main steps when proving an implication of regular formulas $\forall \boldsymbol{x}.\ \boldsymbol{P x} \multimap \boldsymbol{Q x}$: first, one should obtain the P-states of Q; then one should try to build invariants from those P-states. The first step might fail, the second one is decidable. This can be interpreted very naturally in terms of proof-search. The computation of the P-states would correspond to an exhaustive proof-search for $\boldsymbol{P x} \vdash \boldsymbol{Q x}$, only unfolding fixed points and detecting loops. This is visible on the previous example, as well as on the totality of *half*:

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathit{half}\ 0\ H} \qquad \dfrac{\boxed{\dfrac{\dfrac{\dfrac{}{\vdash 0 = 0} \quad \dfrac{\bot}{\vdash sz = 0}}{\mathit{nat}\ y \vdash y = 0}}{\mathit{nat}\ y \vdash sy = s0 \otimes H = 0}} \oplus \boxed{\dfrac{\dfrac{\dfrac{\bot}{\vdash 0 = sZ \otimes \ldots} \quad \dfrac{\dfrac{\infty}{\mathit{nat}\ z \vdash \mathit{half}\ z\ H'}}{\mathit{nat}\ z \vdash sz = sZ \otimes \mathit{half}\ Z\ H'}}{\mathit{nat}\ y \vdash y = sZ \otimes \mathit{half}\ Z\ H'}}{\mathit{nat}\ y \vdash sy = s^2 Z \otimes H = sH' \otimes \mathit{half}\ Z\ H'}}}{\mathit{nat}\ y \vdash \mathit{half}\ (sy)\ H}}{\mathit{nat}\ x \vdash \mathit{half}\ x\ H}}{\mathit{nat}\ x \vdash \exists h.\ \mathit{half}\ x\ h}$$

If that exploration terminates, the information about loops, failed and proved branches can be checked for correctness. This second step, when successful, yields a proof by explicit induction — it might also be possible to produce counter-examples in case of failure. Theorem 2 can thus be read as the possibility to decide the provability of any regular implication, as long as the exhaustive search space is finitely presentable.

5 Conclusion

We have shown that μMALL offers a natural framework for reasoning about automata, in particular about inclusions. Our study lead to the coinductive characterization of inclusion as multi-similarity, and on the proof-theoretical side to an internal completeness result for regular formulas. Finally, our work opened promising avenues for understanding and extending cyclic proof techniques.

An obvious direction for future work is the implementation of the proof-search techniques the we outlined above. But we should also consider extending our results to richer fragments of the logic. A natural next step in that direction would be to study tree automata, and extend regular formulas accordingly. Going significantly further, we have outlined in [1] a way to handle Büchi automata. It is challenging and raises several important problems that are interesting in themselves for the understanding of the field.

Acknowledgments. The author is grateful to Luigi Santocanale, Alwen Tiu and especially Dale Miller for their advices, knowledge and insightful discussions.

References

1. Baelde, D.: A linear approach to the proof-theory of least and greatest fixed points. PhD thesis, Ecole Polytechnique (December 2008)
2. Baelde, D., Gacek, A., Miller, D., Nadathur, G., Tiu, A.: The Bedwyr system for model checking over syntactic expressions. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 391–397. Springer, Heidelberg (2007)
3. Baelde, D., Miller, D.: Least and greatest fixed points in linear logic. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS, vol. 4790, pp. 92–106. Springer, Heidelberg (2007)
4. Brotherston, J.: Cyclic proofs for first-order logic with inductive definitions. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS (LNAI), vol. 3702, pp. 78–92. Springer, Heidelberg (2005)
5. Cleaveland, R.: Tableau-based model checking in the propositional mu-calculus. Acta Informatica 27, 725–747 (1990)
6. Girard, J.-Y.: A fixpoint theorem in linear logic. An email posting to the mailing list linear@cs.stanford.edu (February 1992)
7. McDowell, R., Miller, D.: Cut-elimination for a logic with definitions and induction. Theoretical Computer Science 232, 91–119 (2000)
8. Miller, D., Tiu, A.: A proof theory for generic judgments. ACM Trans. on Computational Logic 6(4), 749–783 (2005)
9. Momigliano, A., Tiu, A.: Induction and co-induction in sequent calculus. In: Berardi, S., Coppo, M., Damiani, F. (eds.) TYPES 2003. LNCS, vol. 3085, pp. 293–308. Springer, Heidelberg (2004)
10. Santocanale, L.: A calculus of circular proofs and its categorical semantics. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 357–371. Springer, Heidelberg (2002)
11. Schroeder-Heister, P.: Rules of definitional reflection. In: Vardi, M. (ed.) Eighth Annual Symposium on Logic in Computer Science, June 1993, pp. 222–232. IEEE Computer Society Press, IEEE, Los Alamitos (1993)
12. Spenger, C., Dams, M.: On the structure of inductive reasoning: Circular and tree-shaped proofs in the μ-calculus. In: Gordon, A.D. (ed.) FOSSACS 2003. LNCS, vol. 2620, pp. 425–440. Springer, Heidelberg (2003)
13. Tiu, A.: Model checking for π-calculus using proof search. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 36–50. Springer, Heidelberg (2005)
14. Tiu, A., Nadathur, G., Miller, D.: Mixing finite success and finite failure in an automated prover. In: Empirically Successful Automated Reasoning in Higher-Order Logics (ESHOL 2005), December 2005, pp. 79–98 (2005)

Decidability for Priorean Linear Time Using a Fixed-Point Labelled Calculus

Bianca Boretti[1] and Sara Negri[2]

[1] Dept. of Electronics and Information, Polytechnic of Milan
[2] Dept. of Philosophy, University of Helsinki

Abstract. A labelled sequent calculus is proposed for Priorean linear time logic, the rules of which reflect a natural closure algorithm derived from the fixed-point properties of the temporal operators. All the rules of the system are finitary, but proofs may contain infinite branches. Soundness and completeness of the calculus are stated with respect to a notion of provability based on a condition on derivation trees: A sequent is provable if and only if no branch leads to a 'fulfilling sequent,' the syntactical counterpart of a countermodel for an invalid sequent. Decidability is proved through a terminating proof search procedure, with an exponential bound to the branches of derivation trees for valid sequents, calculated on the length of the characteristic temporal formula of the endsequent.

1 Introduction

What is commonly known as unary propositional linear time logic (LTL) is the future-oriented reflexive version of Priorean linear time logic: Only the future operators **G**, **F** and **T** are considered in unary LTL, and **G** and **F** have the intuitive meanings of 'it is and will always be the case' and 'it is or will be the case', respectively. LTL is known to be decidable [13]. Decidability has been established by several authors [15], [6], [7] through 2-phase tableau systems: In such systems, after the construction of the tableau graph, a second phase is required in order to check whether every eventuality formula has been satisfied.

In [11] a tableau system has been proposed, in which the termination of proof search can be determined locally, but the system covers only a limited fragment of LTL. In [12] a decision procedure for the whole logic has been achieved through a tableau calculus in which the second phase is incorporated into the rules by annotating sets of formulas with history information. However, this system contains a loop rule which hides a non-local closing condition: In fact, whereas the rules of the system act top-down, the "the result part [...] is synthesized bottom-up (from children to parents)" (p. 286), thus it is necessary to inspect previous nodes in order to verify if there is a loop. In [4] there are local rules with history annotation; Decidability of the calculus is not explicitly stated and would require a similar non-local closing condition in the form of a loop check.

In [2] a labelled sequent calculus G3LT for reflexive Priorean linear time was defined through the method of internalization of the possible world semantics

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 108–122, 2009.

within the syntax of sequent calculi, as developed by the second author in [8,9]. The calculus has all the structural rules admissible, but it requires an infinitary rule to the effect that between any two points there are only finitely many other points. By replacing the infinitary rule with two weaker finitary rules a system for non-standard discrete frames was obtained and a conservativity result for an appropriate fragment of the original calculus proved.

In the present work, a labelled calculus $\mathrm{G3LT}_{cl}$ is defined, the rules of which are justified by a closure algorithm that exploits the fixed-point properties of temporal operators, as proposed for example in [5]. All the rules of the system $\mathrm{G3LT}_{cl}$ are finitary, however, proofs generally require infinite descent in the sense of [3]. Admissibility of cut for $\mathrm{G3LT}_{cl}$ is not established syntactically but as a consequence of completeness. This is unproblematic, because the calculus is conceived as an instrument for establishing decidability of Priorean linear time.

Decidability is proved through a terminating proof search procedure: If a sequent is not a theorem of Priorean linear time logic, then root-first application of the rules of $\mathrm{G3LT}_{cl}$ leads, by the use of labels, to another sequent that supplies an immediate and simple construction of a countermodel. If, on the other hand, we start with a derivable sequent, a finite bound allows to truncate any potentially infinite branch. This establishes at the same time a direct proof of completeness with respect to Kripke semantics. The definition of proofs in $\mathrm{G3LT}_{cl}$ is completely local, and termination is determined with no need of checking previous parts of the derivation because every sequent keeps all the information required.

The calculus $\mathrm{G3LT}_{cl}$ contains also past temporal operators and the decision procedure is given in the strong form of an explicit bound on proof search, although the absence of a global condition on derivations imposes an exponential size on it. We remark, however, that the main purpose of this paper is not to establish decidability, but to illustrate a very general method through its uniform application to linear temporal logic. Although a proof-theoretic approach is followed, the use of labels permits to formalize model-theoretic arguments and to obtain direct proofs of validity and completeness.

The paper is organized as follows: In Section 2 we give the definition of the fixed-point proof system $\mathrm{G3LT}_{cl}$; In Section 3 we identify the proofs in the system; We prove soundness in Section 4 and completeness in Section 5; Decidability through termination of proof search is established in Section 6. For background, and the treatment of Until and Since that have not been included here, we refer to the first author's Ph.D. thesis [1]. For a concise illustration of the general method employed in this work, including the system G3LT, see [2].

2 A Fixed-Point Proof System

The presence of induction constitutes an intrinsic obstacle to the possibility of establishing decidability of Priorean linear time logic through a terminating proof-search procedure. In this paper, we present a labelled calculus $\mathrm{G3LT}_{cl}$ for Priorean linear time. All the rules are finitary, but proofs generally require arguments by infinite descent in the sense of [3]. In a temporal frame for Priorean linear time, between any two points there are only finitely many other points, therefore any model that appeals to an infinite increasing or decreasing sequence of points between two instants can be ignored. The proof-theoretic counterpart of that occurs, for example, when root-first applications of the rules do never realize a future formula $x : \mathbf{F}B$ in the antecedent with a labelled formula $y : B$ and a finite chain $x \prec y_0, \ldots, y_n \prec y$.

A particular class of sequents, which correspond to the syntactic counterparts of countermodels for unprovable purely logical sequents (defined below), is identified and used for giving a sound and complete definition of proofs in G3LT$_{cl}$. Termination of proof search is then obtained thanks to the analogy of the rules of the calculus to the algorithm that produces saturated subsets of formulas.

The basic idea is to formulate a labelled calculus G3LT$_{cl}$ from the fixed-point properties of temporal operators:

$$\begin{array}{ll} \mathbf{G}A \supset\subset \mathbf{T}A \;\&\; \mathbf{TG}A & \qquad \mathbf{F}A \supset\subset \mathbf{T}A \vee \mathbf{TF}A \\ \mathbf{H}A \supset\subset \mathbf{Y}A \;\&\; \mathbf{YH}A & \qquad \mathbf{P}A \supset\subset \mathbf{Y}A \vee \mathbf{YP}A \end{array}$$

In a standard proof of decidability for LTL, as given for example in [13], [14], [5], a countermodel for an invalid sentence is constructed as a relational structure where a saturated set of closure formulas Δ is the immediate successor of a saturated set of closure formulas Γ if $A \in \Delta$ whenever $\mathbf{T}A \in \Gamma$, and a fairness condition is satisfied, namely that all the eventualities of the form $\mathbf{F}A$ are fulfilled at some point. Here the notion of ($\prec$-)saturated label (see Definitions 11, 12) will be defined in order to identify the class of sequents which correspond to countermodels for invalid sequents.

In initial sequents, ϕ is either an atomic formula or a formula prefixed by $\mathbf{T}$ or $\mathbf{Y}$. The propositional rules are identical to those of G3LT in [2]. Repetition of the principal formula in the premisses of $R\mathbf{G}_{cl}$, $L\mathbf{F}_{cl}$, $R\mathbf{H}_{cl}$ and $L\mathbf{P}_{cl}$ is required for the definition of fulfilling sequent (see Definition 19). If the flow of time is linear and unbounded, the next-time operator $\mathbf{T}$ and the previous-time operator $\mathbf{Y}$ satisfy the following, where $x \prec y$ means that x is the immediate predecessor of y (or y the immediate successor of x):

$$x \Vdash \mathbf{T}A \textit{ iff for all } y,\ x \prec y \textit{ implies } y \Vdash A, \textit{ iff for some } y,\ x \prec y \textit{ and } y \Vdash A$$
$$x \Vdash \mathbf{Y}A \textit{ iff for all } y,\ y \prec x \textit{ implies } y \Vdash A, \textit{ iff for some } y,\ y \prec x \textit{ and } y \Vdash A$$

In analogy with the rules for the quantifiers, the universal semantic explanation for $\mathbf{T}$ and $\mathbf{Y}$ would give a variable condition in the right rule, whereas the existential semantic explanation would give a variable condition in the left rule. Because of linearity, both explanations are available, and the rules for $\mathbf{T}$ and $\mathbf{Y}$ can conveniently be formulated in the form given in Table 1, which uses the universal semantic explanation for the left rule and the existential explanation for the right one. Thus, no variable condition is required.

The calculus G3LT$_{cl}$ does not permit syntactic cut elimination. This is because the rules for $\mathbf{T}$ and $\mathbf{Y}$ are given in a non-harmonious way, that is, the left and the right rules are justified by different semantical explanations. However, it is precisely because of this particular choice of rules that the essential properties of G3LT$_{cl}$ hold. Also, we will show that the system without cut is complete, and thus prove that G3LT$_{cl}$ is closed with respect to cut.

The notion of derivability in the calculus G3LT$_{cl}$ is defined in the standard way: A derivation is an initial sequent, or an instance of $L\bot$, or is obtained by an application of a logical or mathematical rule to the derivation(s) concluding its premiss(es). In Section 3 we shall introduce a generalized notion of provability in G3LT$_{cl}$, which admits derivation trees with infinite branches.

Table 1. The rules of the calculus G3LT$_{cl}$

Initial sequents and $L\bot$

$$x : \phi, \Gamma \Rightarrow \Delta, x : \phi \qquad\qquad \frac{}{x : \bot, \Gamma \Rightarrow \Delta}\, {\scriptstyle L\bot}$$

Fixed-point rules

$$\frac{x : \mathbf{T}A, x : \mathbf{TG}A, \Gamma \Rightarrow \Delta}{x : \mathbf{G}A, \Gamma \Rightarrow \Delta}\, {\scriptstyle L\mathbf{G}_{cl}} \qquad \frac{\Gamma \Rightarrow \Delta, x : \mathbf{G}A, x : \mathbf{T}A \quad \Gamma \Rightarrow \Delta, x : \mathbf{G}A, x : \mathbf{TG}A}{\Gamma \Rightarrow \Delta, x : \mathbf{G}A}\, {\scriptstyle R\mathbf{G}_{cl}}$$

$$\frac{x : \mathbf{T}A, x : \mathbf{F}A, \Gamma \Rightarrow \Delta \quad x : \mathbf{TF}A, x : \mathbf{F}A, \Gamma \Rightarrow \Delta}{x : \mathbf{F}A, \Gamma \Rightarrow \Delta}\, {\scriptstyle L\mathbf{F}_{cl}} \qquad \frac{\Gamma \Rightarrow \Delta, x : \mathbf{T}A, x : \mathbf{TF}A}{\Gamma \Rightarrow \Delta, x : \mathbf{F}A}\, {\scriptstyle R\mathbf{F}_{cl}}$$

$$\frac{x : \mathbf{Y}A, x : \mathbf{YH}A, \Gamma \Rightarrow \Delta}{x : \mathbf{H}A, \Gamma \Rightarrow \Delta}\, {\scriptstyle L\mathbf{H}_{cl}} \qquad \frac{\Gamma \Rightarrow \Delta, x : \mathbf{H}A, x : \mathbf{Y}A \quad \Gamma \Rightarrow \Delta, x : \mathbf{H}A, x : \mathbf{YH}A}{\Gamma \Rightarrow \Delta, x : \mathbf{H}A}\, {\scriptstyle R\mathbf{H}_{cl}}$$

$$\frac{x : \mathbf{Y}A, x : \mathbf{P}A, \Gamma \Rightarrow \Delta \quad x : \mathbf{YP}A, x : \mathbf{P}A, \Gamma \Rightarrow \Delta}{x : \mathbf{P}A, \Gamma \Rightarrow \Delta}\, {\scriptstyle L\mathbf{P}_{cl}} \qquad \frac{\Gamma \Rightarrow \Delta, x : \mathbf{Y}A, x : \mathbf{YP}A}{\Gamma \Rightarrow \Delta, x : \mathbf{P}A}\, {\scriptstyle R\mathbf{P}_{cl}}$$

Tomorrow and Yesterday rules

$$\frac{x \prec y, y : A, x : \mathbf{T}A, \Gamma \Rightarrow \Delta}{x \prec y, x : \mathbf{T}A, \Gamma \Rightarrow \Delta}\, {\scriptstyle L\mathbf{T}} \qquad\qquad \frac{x \prec y, \Gamma \Rightarrow \Delta, x : \mathbf{T}A, y : A}{x \prec y, \Gamma \Rightarrow \Delta, x : \mathbf{T}A}\, {\scriptstyle R\mathbf{T}_{cl}}$$

$$\frac{y \prec x, y : A, x : \mathbf{Y}A, \Gamma \Rightarrow \Delta}{y \prec x, x : \mathbf{Y}A, \Gamma \Rightarrow \Delta}\, {\scriptstyle L\mathbf{Y}} \qquad\qquad \frac{y \prec x, \Gamma \Rightarrow \Delta, x : \mathbf{Y}A, y : A}{y \prec x, \Gamma \Rightarrow \Delta, x : \mathbf{Y}A}\, {\scriptstyle R\mathbf{Y}_{cl}}$$

Mathematical rules:

$$\frac{y \prec x, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\, L\text{-}Ser \qquad\qquad \frac{x \prec y, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\, R\text{-}Ser$$

Rules *L-Ser* and *R-Ser* have the condition that y is not in the conclusion.

A rule is height-preserving admissible if, whenever its premiss(es) is (are) derivable, also its conclusion is derivable with the same bound on the derivation height; A rule is height-preserving invertible if, whenever its conclusion is derivable, also its premiss(es) is (are) derivable with the same bound on the derivation height. The proofs of the following structural results are detailed in [1].

Proposition 1. *Substitution of labels is height-preserving admissible in* G3LT$_{cl}$. *All the rules of* G3LT$_{cl}$ *are height-preserving invertible. Weakening and contraction are height-preserving admissible in* G3LT$_{cl}$.

Definition 2. *In an instance of rule* R-Ser *(resp.* L-Ser*) with active formula $x \prec y$ (resp. $y \prec x$), the label x is called* side label.

Lemma 3. *A derivation in* G3LT$_{cl}$ *can be transformed into a derivation with all instances of* R-Ser *and* L-Ser *applied on side labels that appear in the conclusion of the rule.*

Root-first proof search can, without loss of generality, be restricted to *minimal derivations*, that is, derivations which cannot be shortened through height-preserving admissibility of contraction or other local modifications: In particular, applications of rules that produce duplications of atoms when read from conclusion to premisses can be dispensed with by height-preserving admissibility of contraction. The same holds if a duplication occurs modulo fresh replacement of eigenvariables, so we have:

Lemma 4. *In a minimal derivation in* G3LT$_{cl}$*, rule* R-Ser *(resp.* L-Ser*) need not be applied on a relational atom $x \prec y$ (resp. $y \prec x$) if its conclusion contains an atom $x \prec z$ (resp. $z \prec x$) in the antecedent.*

Lemma 5. *The rules* L-Ser *and* R-Ser *permute up with respect to all the rules of* G3LT$_{cl}$ *in case their eigenvariable is not contained in the active formula(s) of the latter.*

Lemma 6. *On any branch of a minimal derivation in* G3LT$_{cl}$*, a given temporal rule with the repetition of the principal formula(s) in the premiss(es) need not be applied more than once on the same formulas.*

A *purely logical sequent* is a sequent that contains no relational atoms and in which every formula is labelled by the same variable. Every purely logical sequent $\Gamma \Rightarrow \Delta$ with all its formulas labelled by x corresponds to a characteristic formula $\wedge\Gamma^x \supset \vee\Delta^x$, where $\Gamma^x = \{A \mid x : A \in \Gamma\}$, and similarly Δ^x. With this identification, the rules of the system G3LT$_{cl}$, read root first, correspond to the algorithm for producing the saturated subsets of closure formulas from a given formula.

Definition 7. *The set $cl(A)$ of* closure formulas *of a formula A is defined inductively as follows:*

- *$B \in cl(A)$ for every subformula B of A;*
- *$\mathbf{T}B$ and $\mathbf{TG}B \in cl(A)$ if $\mathbf{G}B \in cl(A)$;*
- *$\mathbf{T}B$ and $\mathbf{TF}B \in cl(A)$ if $\mathbf{F}B \in cl(A)$;*
- *$\mathbf{Y}B$ and $\mathbf{YH}B \in cl(A)$ if $\mathbf{H}B \in cl(A)$;*
- *$\mathbf{Y}B$ and $\mathbf{YP}B \in cl(A)$ if $\mathbf{H}B \in cl(A)$.*

Lemma 8. *Let $|A|$ be the number of subformulas of A. The cardinality of $cl(A)$ is linearly bounded by $|A|$, namely $|cl(A)| \leq 3 \cdot |A|$.*

Proof. By induction on the length of A.

Corollary 9. *The number of subsets of $cl(A)$ is at most $2^{3|A|}$.*

The definition of a satured set of formulas is an extension of the classical definition, obtained for the temporal modalities from their fixed-point properties.

Definition 10. *A set S of formulas is* saturated *if the following conditions are satisfied:*

- $\bot$ *is not in* S*;*
- *For every formula* B*, it is not possible that both* B *and* $\neg B$ *are in* S*;*
- $\neg\neg B$ *in* S *implies that* B *is in* S*;*
- $B \& C$ *in* S *implies that both* B *and* C *are in* S*;*
- $\neg(B \& C)$ *in* S *implies that either* $\neg B$ *or* $\neg C$ *is in* S*;*
- $B \vee C$ *in* S *implies that* B *or* C *is in* S*;*
- $\neg(B \vee C)$ *in* S *implies both* $\neg B$ *and* $\neg C$ *are in* S*;*
- $B \supset C$ *in* S *implies that either* $\neg B$ *or* C *is in* S*;*
- $\neg(B \supset C)$ *in* S *implies that both* B *and* $\neg C$ *are in* S*;*
- $\mathbf{G}B$ *in* S *implies that both* $\mathbf{T}B$ *and* $\mathbf{TG}B$ *are in* S*;*
- $\neg\mathbf{G}B$ *in* S *implies that either* $\neg\mathbf{T}B$ *or* $\neg\mathbf{TG}B$ *is in* S*;*
- $\mathbf{F}B$ *in* S *implies that* $\mathbf{T}B$ *or* $\mathbf{TF}B$ *is in* S*;*
- $\neg\mathbf{F}B$ *in* S *implies that both* $\neg\mathbf{T}B$ *and* $\neg\mathbf{TF}B$ *are in* S*;*
- $\mathbf{H}B$ *in* S *implies that both* $\mathbf{Y}B$ *and* $\mathbf{YH}B$ *are in* S*;*
- $\neg\mathbf{H}B$ *in* S *implies that either* $\neg\mathbf{Y}B$ *or* $\neg\mathbf{YH}B$ *is in* S*;*
- $\mathbf{P}B$ *in* S *implies that* $\mathbf{Y}B$ *or* $\mathbf{YP}B$ *is in* S*;*
- $\neg\mathbf{P}B$ *in* S *implies that both* $\neg\mathbf{Y}B$ *and* $\neg\mathbf{YP}B$ *are in* S*.*

The notions of saturated and $\prec$-saturated label in a sequent are then given as follows:

Definition 11. *A label* x *in* $\Gamma \Rightarrow \Delta$ *is* saturated *if the set* $\Gamma^x \cup \overline{\Delta^x}$ *is saturated, where* $\Gamma^x\{B \mid x : B \in \Gamma\}$*,* $\overline{\Delta^x} = \{\overline{B} \mid x : B \in \Delta\}$*, and* $\overline{B} \equiv \neg B$ *if* $B \neq \neg C$*,* $\overline{B} \equiv C$ *otherwise.*

Definition 12. *A label* x *in* $\Gamma \Rightarrow \Delta$ *is* $\prec$-saturated *if it is saturated and:*

- $x : \mathbf{T}B$ *in* $\Gamma(\Delta)$ *implies that, if* $x \prec y$ *is in* Γ*, then* $y : B$ *is in* $\Gamma(\Delta)$*;*
- $x : \mathbf{Y}B$ *in* $\Gamma(\Delta)$ *implies that, if* $y \prec x$ *is in* Γ*, then* $y : B$ *is in* $\Gamma(\Delta)$*.*

3 Proofs in G3LT$_{cl}$

In this Section we shall define the proofs in G3LT$_{cl}$ through the identification of a particular class of sequents, which can be considered finite syntactical counterparts of countermodels for invalid sequents.

Given a purely logical sequent $\Gamma \Rightarrow \Delta$, we start a proof search by applying root-first the rules of G3LT$_{cl}$ for the propositional connectives and for **G**, **F**, **H**, and **P**, whenever possible. When x becomes saturated, we apply once the rules *R-Ser* and *L-Ser* with side label x, thus introducing new labels y and y' and the accessibility relations $x \prec y$ and $y' \prec x$. By Lemma 5 we are allowed to postpone the application of the rules for seriality until no more logical rule can be applied, and by Lemmas 3 and 4 we do not need to apply a seriality rule with side label z, if z is not a label in the sequent or the antecedent already contains an atom $z \prec z'$ (resp. $z' \prec z$). Next, we apply the rules $L\mathbf{T}$ and $R\mathbf{T}_{cl}$ (resp. $L\mathbf{Y}$

and $R\mathbf{Y}_{cl}$) on the formulas with $\mathbf{T}$ (resp. $\mathbf{Y}$) as their outermost operator until x becomes $\prec$-saturated. Note that by Lemma 6, we need not apply more than once a temporal rule on the same principal formula(s). We repeat the procedure with the formulas marked by y and y'. We continue as before with all the labels possibly introduced by *R-Ser* and *L-Ser*, and so on. This procedure motivates the following definition:

Definition 13. *A* pre-proof *of a purely logical sequent in* G3LT$_{cl}$ *is a (possibly infinite) tree obtained by applying root-first the logical and mathematical rules of the calculus, whenever possible.*

Before giving the definition of a proof in G3LT$_{cl}$, we need some preliminary notions. We shall construct the syntactic counterpart of a countermodel from a failed proof search and therefore define syntactic entities through their correspondence to a Kripke model for Priorean linear time.

Definition 14. *A* discrete linear temporal frame $\mathcal{F} = (\mathcal{K}, \prec^{\mathcal{K}}, <^{\mathcal{K}})$ *is a linearly ordered set, with the order relation* $<^{\mathcal{K}}$ *defined as the transitive closure of the immediate successor relation* $\prec^{\mathcal{K}}$*, functional and unbounded in both directions.*

Definition 15. *Let* $\mathcal{F} = (\mathcal{K}, \prec^{\mathcal{K}}, <^{\mathcal{K}})$ *be a discrete linear temporal frame. An* evaluation *of atomic formulas in a frame is a map* $\mathcal{V} : AtFrm \to \mathcal{P}(\mathcal{K})$*, assigning to any atom* P *the set of instants in which* P *holds. The standard notation for* $k \in \mathcal{V}(P)$ *is* $k \Vdash P$*. Evaluations are extended to arbitrary formulas by the following inductive clauses:*

For all $k \in \mathcal{K}$*, it is not the case that* $k \Vdash \bot$ *(abbr.* $k \nVdash \bot$*);*
$k \Vdash A\&B$ *if* $k \Vdash A$ *and* $k \Vdash B$*;*
$k \Vdash A \vee B$ *if* $k \Vdash A$ *or* $k \Vdash B$*;*
$k \Vdash A \supset B$ *if* $k \Vdash A$ *implies* $k \Vdash B$*;*
$k \Vdash \mathbf{G}A$ *(resp.* $k \Vdash \mathbf{H}A$*) if for all* k', $k <^{\mathcal{K}} k'$ *(resp.* $k' <^{\mathcal{K}} k$*) implies* $k' \Vdash A$*;*
$k \Vdash \mathbf{F}A$ *(resp.* $k \Vdash \mathbf{P}A$*) if for some* k', $k <^{\mathcal{K}} k'$ *(resp.* $k' <^{\mathcal{K}} k$*) and* $k' \Vdash A$
$k \Vdash \mathbf{T}A$ *(resp.* $k \Vdash \mathbf{Y}A$*) if for all* k', $k \prec^{\mathcal{K}} k'$ *(resp.* $k' \prec^{\mathcal{K}} k$*) implies* $k' \Vdash A$

The definition of evaluation of formulas justifies the notion of interpretation of the labels of a sequent and of validity for labelled formulas and relational atoms in a discrete linear temporal frame:

Definition 16. *Let* $\mathcal{F} = (\mathcal{K}, \prec^{\mathcal{K}}, <^{\mathcal{K}})$ *be a linear discrete frame with accessibility relations* $<^{\mathcal{K}}$ *and* $\prec^{\mathcal{K}}$*. Let* W *be the set of labels used in the derivation of the sequent* $\Gamma \Rightarrow \Delta$ *in* G3LT$_{cl}$*. An* interpretation *of the labels from* W *in* $\mathcal{K}$ *is a function* $[\![\cdot]\!] : W \to \mathcal{K}$*. A* countermodel *to* $\Gamma \Rightarrow \Delta$ *is a discrete linear temporal frame* $(\mathcal{K}, \prec^{\mathcal{K}}, <^{\mathcal{K}})$ *together with an interpretation and an evaluation that validates all the formulas and relational atoms in* Γ *and no formula in* Δ*; Namely, for all labelled formulas* $z : A$ *and relational atoms* $x \prec y$ *in the antecedent,* $[\![z]\!] \Vdash A$ *and* $[\![x]\!] \prec^{\mathcal{K}} [\![y]\!]$ *but for no* $w : B$ *in the succedent* $[\![w]\!] \Vdash B$*.*

The semantic explanations for the possibility-like temporal operators $\mathbf{F}$, $\mathbf{P}$ and the definition of the order relation $<^{\mathcal{K}}$ as the transitive closure of the immediate successor relation $\prec^{\mathcal{K}}$ justify the following notion of future and past witness. We use the standard symbol for syntactic identity "$x \equiv y$" to denote that x and y are the same syntactic object.

Definition 17. *Given a labelled formula $z : \mathbf{F}B$ in the antecedent of a sequent $\Gamma \Rightarrow \Delta$ (resp. $z : \mathbf{G}B$ in the succedent), we say that a label z' is a* future witness *for $z : \mathbf{F}B$ (resp. $z : \mathbf{G}B$) if $z' : B$ is in Γ (resp. $z' : B$ is in Δ) and the relational atoms $z \prec z_0, \ldots, z_{n-1} \prec z_n \equiv z'$ are in Γ for some n.*
Given a labelled formula $z : \mathbf{P}B$ in the antecedent of a sequent $\Gamma \Rightarrow \Delta$ (resp. $z : \mathbf{H}B$ in the succedent), we say that a label z' is a past witness *for $z : \mathbf{P}B$ (resp. $z : \mathbf{H}B$) if $z' : B$ is in Γ (resp. $z' : B$ is in Δ) and the relational atoms $z' \prec z_0, \ldots, z_{n-1} \prec z_n \equiv z$ are in Γ for some n.*

In the syntactic object that corresponds to a Priorean linear time model, we have to ensure that every possibility-like formulas is realized by some label:

Definition 18. *A chain $z_i \prec z_{i+1}, \ldots, z_{j-1} \prec z_j$ (with $j \geq i+1$) in a sequent $\Gamma \Rightarrow \Delta$ is a* future loop *if z_j marks exactly the same formulas as the label z_i and, for every labelled formula $z_q : \mathbf{F}B$ in Γ (resp. $z_q : \mathbf{G}B$ in Δ) with $i \leq q \leq j$, there exists z_k such that $i \leq k \leq j$ and $z_k : B$ is in Γ (resp. in Δ). We call z_j the* future looping label *with respect to z_i*
A chain $z_i \prec z_{i+1}, \ldots, z_{j-1} \prec z_j$ (with $j \geq i+1$) in a sequent $\Gamma \Rightarrow \Delta$ is a past loop *if z_i marks exactly the same formulas as the label z_j and, for every labelled formula $z_q : \mathbf{P}B$ in Γ (resp. $z_q : \mathbf{H}B$ in Δ) with $i \leq q \leq j$, there exists some variable z_k such that $i \leq k \leq j$ and $z_k : B$ is in Γ (resp. in Δ). We call z_i the* past looping label *with respect to z_j.*

A root-first proof search succeeds when a derivation is found, namely all the leaves of the derivation tree are initial sequents or instances of $L\bot$. However, a failed proof search does not in general ensure that an endsequent $\Gamma \Rightarrow \Delta$ is invalid unless a countermodel can be constructed from it. Here comes into play the notion of fulfilling sequent for a purely logical sequent $\Gamma \Rightarrow \Delta$:

Definition 19. *Let the sequent $\Gamma^* \Rightarrow \Delta^*$ be obtained by root-first proof search from the purely logical sequent $\Gamma \Rightarrow \Delta$ (with all its formulas labelled by x). Then, $\Gamma^* \Rightarrow \Delta^*$ is a* fulfilling sequent *if the following conditions are satisfied:*

(i) Every label in it is $\prec$-saturated;
(ii) It contains a chain of relational atoms $z_{-m} \prec z_{-(m-1)}, \ldots, z_{-1} \prec z_0 \equiv x$, $z_0 \prec z_1, \ldots, z_{n-1} \prec z_n$, such that for some i with $-m < i \leq 0$ the subchain $z_{-m} \prec z_{-(m-1)}, \ldots, z_{i-1} \prec z_i$ is a past loop, and for some j with $0 \leq j < n$, the subchain $z_j \prec z_{j+1}, \ldots, z_{n-1} \prec z_n$ is a future loop;
(iii) Every labelled formula $z : \mathbf{F}B$ in Γ^* (resp. $z : \mathbf{G}B$ in Δ^*) is either witnessed by a future witness label z', or has z inside a future loop;
(iv) Every labelled formula $z : \mathbf{P}B$ in Γ^* (resp. $z : \mathbf{H}B$ in Δ^*) is either witnessed by a past witness label z', or has z inside a past loop.

Intuitively, a fulfilling sequent corresponds to a structure constituted by a (possibly empty) linear chain with two simple loops at the ends, with the left and the right loop obtained by identifying the first and the last label of the past and of the future loop, respectively.

In Section 4 we shall prove that, given a model for Priorean linear time, it is possible to extract the corresponding fulfilling sequent, and in Section 5 we shall show how to linearize the future and the past loop in order to obtain an appropriate model.

Proposition 20. *Let $\Gamma' \Rightarrow \Delta'$ be obtained by applying root-first the rules of* G3LT$_{cl}$ *from the purely logical sequent $\Gamma \Rightarrow \Delta$ with x as the uniform label that marks all the formulas in the latter. Then $\Gamma' \Rightarrow \Delta'$ contains a unique chain $z_{-m} \prec z_{-(m-1)}, \ldots, z_{-1} \prec z_0 \equiv x$, $z_0 \prec z_1, \ldots, z_{n-1} \prec z_n$ with z_i different from z_j for $i \neq j$.*

Proof. Since the root sequent $\Gamma \Rightarrow \Delta$ is purely logical, the result follows by Lemmas 3, 4 and the fact that only seriality rules can introduce relational atoms.

While searching for a fulfilling sequent, we want to find one as small as possible. Therefore we should try to avoid useless circles, namely those exploring instants reachable as well through a more direct path. This motivates the following definition:

Definition 21. *Let $\Gamma' \Rightarrow \Delta'$ be obtained by applying root-first the rules of* G3LT$_{cl}$ *from the purely logical sequent $\Gamma \Rightarrow \Delta$ with x as the uniform label that marks all the formulas in the latter. A chain $y_0 \prec y_1, \ldots, y_{n-1} \prec y_n$ (resp. $y_{-n} \prec y_{-(n-1)}, \ldots, y_{-1} \prec y_0$) with $y_0 \equiv x$ in $\Gamma' \Rightarrow \Delta'$ is* roundabout *if it contains labels y_i, y_j with $0 \leq i < j \leq n$ such that y_i and y_j mark the same formulas, $y_i \prec y_{i+1}, \ldots, y_{j-1} \prec y_j$ is not the future loop (resp. the past loop) and either $j = i+1$ or for every y_k with $i < k < j$ there exists some y_l such that $l > j$ (resp. $l < i$) and y_k and y_l mark the same formulas. We say that the subchain $y_i \prec y_{i+1}, \ldots, y_{j-1} \prec y_j$ is* dispensable. *A fulfilling sequent is* reduced *if it does not contain dispensable subchains.*

Note that by Definition 21 a chain can be roundabout also in the case that y_i and y_j mark no formulas.

Theorem 22. *If a proof search for a purely logical sequent $\Gamma \Rightarrow \Delta$ (with all its formulas labelled by x) leads to a fulfilling sequent $\Gamma^* \Rightarrow \Delta^*$, then it also leads to a reduced fulfilling sequent.*

Proof. (*Sketch*) Note that for every label z introduced by *R-Ser* (resp. *L-Ser*) a labelled formula $z : C$ in $\Gamma^* \Rightarrow \Delta^*$ either is introduced by applying root-first the rules $L\mathbf{T}$ and $R\mathbf{T}_{cl}$ (resp. $L\mathbf{Y}$ and $R\mathbf{Y}_{cl}$) or is the result of root-first application of the other rules on a formula introduced in the former way. If the chain $z_0 \prec z_1, \ldots, z_{n-1} \prec z_n$ with $x \equiv z_0$ contains a dispensable subchain $z_i \prec z_{i+1}, \ldots, z_{j-1} \prec z_j$, then the labels z_i and z_j mark the same formulas; Therefore $z_{j+1} : B$ is introduced by $L\mathbf{T}$ (resp. $R\mathbf{T}_{cl}$) with principal formulas

$z_j \prec z_{j+1}, z_j : \mathbf{T}B$ iff $z_{i+1} : B$ can be introduced by $L\mathbf{T}$ (resp. $R\mathbf{T}_{cl}$) with principal formulas $z_i \prec z_{i+1}, z_i : \mathbf{T}B$. Given a set of formulas marked by a label z, the rules of G3LT$_{cl}$ explore different subsets of closure formulas that possibly $\prec$-saturate z: While applying root-first the rules of G3LT$_{cl}$ we have to continue along the branch in which the label z_{i+1} is $\prec$-saturated by the same subset of closure formulas that $\prec$-saturates z_{j+1} in the original fulfilling sequent. By choosing the appropriate premiss of a branching rule whenever a roundabout chain is met, we finally reach the desired reduced fulfilling sequent.

Definition 23. *A pre-proof of a purely logical sequent is a* proof *if no branch in it leads to a fulfilling sequent. A sequent is* provable *if there is a proof for it.*

Every G3LT$_{cl}$ derivation is a G3LT$_{cl}$ proof, but the converse does not hold. Observe that, contrary to the definition of proof in cyclic calculi for induction and infinite descent of [3], our definition in G3LT$_{cl}$ is completely local, i.e. there is no need of checking previous parts of the tree: At any step of the proof search we simply have to consider the sequents introduced by root-first application of the rules and check if they are initial sequents, fulfilling sequents, or neither.

4 Soundness

Soundness for G3LT$_{cl}$ cannot be proved simply by showing that the initial sequents and the rules of the system are sound because, by Definition 23, proofs in G3LT$_{cl}$ can contain infinitely long branches. Therefore, we prove soundness by contraposition: If there exists a countermodel for $\Gamma \Rightarrow \Delta$, then the corresponding proof search should contain a fulfilling sequent and so $\Gamma \Rightarrow \Delta$ is unprovable in G3LT$_{cl}$. Thus, the absence of a fulfilling sequent in a derivation tree is a global soundness condition for a proof.

Some preliminary results concerning standard models are needed: We have to prove that, given a countermodel $\mathcal{M}$ for A, it is possible to extract a fulfilling sequent all the labels of which mark $\prec$-saturated sets of closure formulas of A. The lemmas below show how to construct a future and a past loop from $\mathcal{M}$. In the following, we write $s \leqslant^{\mathcal{K}} s'$ if $s = s'$ or $s <^{\mathcal{K}} s'$ in a model $\mathcal{M} = (\mathcal{K}, \prec^{\mathcal{K}}, <^{\mathcal{K}}, \Vdash)$.

Lemma 24. *Let $\mathcal{M} = (\mathcal{K}, \prec^{\mathcal{K}}, <^{\mathcal{K}}, \Vdash)$ be a model for Priorean linear time and suppose that, for some instant w, $w \nVdash A$. Then for some s such that $w \leqslant^{\mathcal{K}} s$, there exists s' such that $s <^{\mathcal{K}} s'$, s and s' satisfy the same subset $H \subseteq cl(A)$, and for every t if $s \leqslant^{\mathcal{K}} t \leqslant^{\mathcal{K}} s'$ and $t \Vdash \mathbf{F}B$ and $\mathbf{F}B \in cl(A)$ (resp. $t \nVdash \mathbf{G}B$ and $\mathbf{G}B \in cl(A)$) there exists u such that $s \leqslant^{\mathcal{K}} u \leqslant^{\mathcal{K}} s'$ and $u \Vdash B$ (resp. $u \nVdash B$).*

Proof. Since every model for Priorean linear time is isomorphic to the integers, there are infinitely many instants greater than w. However, by Corollary 9, there are only $2^{3|A|}$ subsets of $cl(A)$. By an application of Ramsey's Theorem, for some instant(s) greater than w there exist infinitely many instants satisfying the same subset H of closure formulas of A. Let s be the first instant of the infinite set of instants $s_0 <^{\mathcal{K}} s_1 <^{\mathcal{K}} s_2 <^{\mathcal{K}} s_3 <^{\mathcal{K}} \ldots$ all satisfying the same subset $H \subseteq cl(A)$ and such that $w \leqslant^{\mathcal{K}} s$. Let $s \leqslant^{\mathcal{K}} t$ and $t \Vdash \mathbf{F}B$ and $\mathbf{F}B \in cl(A)$ (resp. $t \nVdash \mathbf{G}B$ and $\mathbf{G}B \in cl(A)$). If there exists a u such that $u \Vdash B$ and $s \leqslant^{\mathcal{K}} u \leqslant^{\mathcal{K}} t$, we are done. Otherwise, since $t \Vdash \mathbf{F}B$ (resp. $t \nVdash \mathbf{G}B$), there exists some u such that $t <^{\mathcal{K}} u$ and $u \Vdash B$ (resp. $u \nVdash B$). Since, by hypothesis, there are infinitely many instants greater than s satisfying H, but u can be reached from t by finitely many iterations of the relation $\prec^{\mathcal{K}}$, for some $i = 1, 2, \ldots$, we have $s <^{\mathcal{K}} u \leqslant^{\mathcal{K}} s_i$. For every i there are only finitely many closure formulas of A of the form $\mathbf{F}B$ (resp. $\mathbf{G}B$) validated (resp. invalidated) by an instant t such that $s \leqslant^{\mathcal{K}} t \leqslant^{\mathcal{K}} s_i$, and for every such t we can find a k and a u such that $s \leqslant^{\mathcal{K}} u \leqslant^{\mathcal{K}} s_{i+k}$ and $u \Vdash B$ (resp. $u \nVdash B$). Since the set of closure formulas of A is finite, the process eventually ends with the determination of a s' such that $s <^{\mathcal{K}} s'$ and for every t if $s \leqslant^{\mathcal{K}} t \leqslant^{\mathcal{K}} s'$ and $t \Vdash \mathbf{F}B$ and $\mathbf{F}B \in cl(A)$ (resp. $t \nVdash \mathbf{G}B$ and $\mathbf{G}B \in cl(A)$) there exists u such that $s \leqslant^{\mathcal{K}} u \leqslant^{\mathcal{K}} s'$ and $u \Vdash B$ (resp. $u \nVdash B$).

Lemma 25. *Let $\mathcal{M} = (\mathcal{K}, \prec^{\mathcal{K}}, <^{\mathcal{K}}, \Vdash)$ be a model for Priorean linear time such that for some instant w, $w \nVdash A$. Then for some instant s such that $s \leqslant^{\mathcal{K}} w$, there exists s' such that $s' <^{\mathcal{K}} s$, s and s' satisfy the same subset $H \subset cl(A)$ and for every t if $s' \leqslant^{\mathcal{K}} t \leqslant^{\mathcal{K}} s$ and $t \Vdash \mathbf{P}B$ and $\mathbf{P}B \in cl(A)$ (resp. $t \nVdash \mathbf{H}B$ and $\mathbf{H}B \in cl(A)$) there exists u such that $s' \leqslant^{\mathcal{K}} u \leqslant^{\mathcal{K}} s$ and $u \Vdash B$ (resp. $u \nVdash B$).*

Proof. Analogous to the proof of Lemma 24.

Lemma 26. *All the rules of* G3LT$_{cl}$ *are sound.*

Proof. The case of the initial sequents and the propositional rules is straightforward. The rules for **G**, **F**, **H** and **P** are sound by definition, since they are justified by their fixed-point interpretations. Similarly, the rules for **T** and **Y** are justified by their semantic explanations, and the mathematical rules correspond to the frame properties of left and right seriality for $\prec$.

Theorem 27. *If a purely logical sequent $\Gamma \Rightarrow \Delta$ (with all its formulas labelled by x) has a countermodel, then it is not provable in* G3LT$_{cl}$.

Proof. Let us suppose that there exists a countermodel $\mathcal{M} = (\mathcal{K}, \prec^{\mathcal{K}}, <^{\mathcal{K}}, \Vdash)$ for the purely logical sequent $\Gamma \Rightarrow \Delta$, namely there exists $w \in \mathcal{K}$ such that $[\![x]\!] = w$ and $w \nVdash \wedge\Gamma^x \supset \vee\Delta^x$. By Lemma 26, every countermodel for the conclusion of any of the rules of G3LT$_{cl}$ is a countermodel for (at least one of) the premiss(es). By choosing the appropriate branch we eventually find a sequent with a chain

$$z_{-m} \prec z_{-(m-1)}, \ldots, z_{-1} \prec z_0 \equiv x,\ z_0 \prec z_1, \ldots, z_{n-1} \prec z_n$$

every label of which matches an instant in the corresponding position in $\mathcal{M}$. To show that this sequent is a fulfilling sequent for $\Gamma \Rightarrow \Delta$, we have to check that the conditions of Definition 19 are satisfied:

(i) By induction on the length of formulas, it is easy to prove that evey label z appearing in the tree can be $\prec$-saturated by applying the rules root-first;

(ii) The presence of a future and a past loop follows from Lemmas 24 and 25, and the fact that we can go on applying right and left seriality rules and introduce new labels until the conditions of the lemmas are satisfied;

(iii) If the formula $z : \mathbf{F}B$ (resp. $z : \mathbf{G}B$) is in the antecedent (resp. succedent), then $[\![z]\!] \Vdash \mathbf{F}B$ (resp. $[\![z]\!] \nVdash \mathbf{G}B$). Therefore, either there exists an instant s such that $[\![z]\!] <^{\mathcal{K}} s$ and $s \Vdash B$ (resp. $s \nVdash B$), and for some z', $[\![z']\!] = s$ and z' is the future witness of $z : \mathbf{F}B$ (resp. $z : \mathbf{G}B$), or $[\![z]\!]$ falls under the conditions of Lemma 24, and thus z is inside a future loop;

(iv) If the formula formula $z : \mathbf{P}B$ (resp. $z : \mathbf{H}B$) is in the antecedent (resp. succedent), then $[\![z]\!] \Vdash \mathbf{P}B$ (resp. $[\![z]\!] \nVdash \mathbf{H}B$). Therefore, either there exists an instant s such that $s <^{\mathcal{K}} [\![z]\!]$ and $s \Vdash B$ (resp. $s \nVdash B$), and for some z', $[\![z']\!] = s$ and z' is the past witness of $z : \mathbf{P}B$ (resp. $z : \mathbf{H}B$), or $[\![z]\!]$ falls under the conditions of Lemma 25, and so z is inside a past loop.

5 Completeness

Completeness is also proved by contraposition: If $\Gamma \Rightarrow \Delta$ is not provable in G3LT$_{cl}$, i.e. if the root-first proof search leads to a fulfilling sequent, then a countermodel for $\Gamma \Rightarrow \Delta$ can be constructed. Our completeness result follows the method in [10]. However, the definition of fulfilling sequents allows to consider only finite objects, and not (possibly) infinite reduction tree; Furthermore, the presence of the fixed-point rules for the temporal operators requires additional work in proving the inductive steps for temporal formulas, since we cannot appeal directly to the semantic explanations for the corresponding operators.

Let us consider the standard frame $\mathcal{F} = (\mathcal{K}, \prec^{\mathcal{K}}, <^{\mathcal{K}})$ for Priorean linear time, with $\mathcal{K} = \{s_i \mid i \in \mathbb{Z}\}$, $s_i \prec^{\mathcal{K}} s_{i+1}$ and $s_i <^{\mathcal{K}} s_j$ for $i < j$. Given a fulfilling sequent $\Gamma^* \Rightarrow \Delta^*$ for the purely logical sequent $\Gamma \Rightarrow \Delta$, we construct a countermodel $\mathcal{M}$ by defining an appropriate interpretation for the set of labels in $\Gamma^* \Rightarrow \Delta^*$ into the domain $\mathcal{K}$ as follows: We put $[\![x]\!] = s_0$ if x is the label that marks all the formulas in $\Gamma \Rightarrow \Delta$, and for every label z if the relational atoms $x \equiv z_0 \prec z_1, \ldots, z_{n-1} \prec z_n \equiv z$ are in Γ, we put $[\![z]\!] = s_n$. Analogously, if $z \equiv z_{-n} \prec z_{-(n-1)}, \ldots, z_{-1} \prec z_0 \equiv x$ are in Γ, we put $[\![z]\!] = s_{-n}$. We evaluate the atomic formulas by putting $[\![z]\!] \Vdash P$ if $z : P$ is in Γ^* and $[\![z]\!] \nVdash P$ if $z : P$ is in Δ^*. Furthermore, if z_{n+l} is the future looping label with respect to z_n, $[\![z_{n+l}]\!] = s_{n+l}$ and $[\![z_n]\!] = s_n$, then for every instant $s_{n+m\cdot l+q}$ (with $m \geq 0$ and $0 \leq q \leq l-1$) we put $s_{n+m\cdot l+q} \Vdash P$ if $z_{n+q} : P$ is in Γ^* and $s_{n+m\cdot l+q} \nVdash P$ if $z_{n+q} : P$ is in Δ^*. Analogously, if $z_{-(n+l)}$ is the past looping label with respect to z_{-n}, $[\![z_{-(n+l)}]\!] = s_{-(n+l)}$ and $[\![z_{-n}]\!] = s_{-n}$, then for every instant $s_{-(n+m\cdot l+q)}$ (with $m \geq 0$ and $0 \leq q \leq l-1$) we put $s_{-(n+m\cdot l+q)} \Vdash P$ if $z_{-(n+q)} : P$ is in Γ^* and $s_{-(n+m\cdot l+q)} \nVdash P$ if $z_{-(n+q)} : P$ is in Δ^*.

Lemma 28. *$\mathcal{M}$ is a countermodel for $\Gamma^* \Rightarrow \Delta^*$.*

Proof. By definition, if $z \prec z'$ is in Γ^*, then $[\![z]\!] \prec^{\mathcal{K}} [\![z']\!]$. We have to show that, for arbitrary formulas B, if $z : B$ is in Γ^*, then $[\![z]\!] \Vdash B$, and if $z : B$ is in Δ^*, then $[\![z]\!] \nVdash B$. We proceed by induction on the length of the formula B. If B is an atomic formula P and $z : P$ is in Γ^*, then $[\![z]\!] \Vdash P$ by construction. If $z : P$ is in Δ^*, then $[\![z]\!] \nVdash P$ by construction. Since z is $\prec$-saturated, $z : P$ cannot be both in Γ^* and in Δ^*. If $B \equiv \bot$, then it cannot be in Γ^* by definition of fulfilling sequent. If $z : \bot$ is in Δ^*, then $[\![z]\!] \nVdash \bot$ by Definition 15. The case of propositional connectives is straightforward. We consider in detail only the cases of $B \equiv \mathbf{T}C$ and $B \equiv \mathbf{G}C$, all the other cases being analogous.

If $B \equiv \mathbf{T}C$ and $z : \mathbf{T}C$ is in $\mathit{\Gamma}^*$ (resp. $\mathit{\Delta}^*$), then we have two cases: (i) If the label z is not the future looping label z_f, then it is connected to it by a chain $z \equiv z_{n+l-i} \prec z_{n+l-(i-1)}, \ldots, z_{n+l-1} \prec z_{n+l} \equiv z_f$ and, since the label z_{n+l-i} is $\prec$-saturated, we have $z_{n+l-(i-1)} : C$ in $\mathit{\Gamma}^*$ (resp. $\mathit{\Delta}^*$). Therefore, by construction, we have $[\![z_{n+l-i}]\!] \prec^{\mathcal{K}} [\![z_{n+l-(i-1)}]\!]$ and by inductive hypothesis $[\![z_{n+l-(i-1)}]\!] \Vdash C$ (resp. $[\![z_{n+l-(i-1)}]\!] \nVdash C$). So $[\![z]\!] \Vdash \mathbf{T}C$ (resp. $[\![z]\!] \nVdash \mathbf{T}C$). (ii) If z is the future looping label, then by definition for no label z' the atom $z \prec z'$ is in $\mathit{\Gamma}^*$. However, we have some label z_n such that $x \equiv z_0 \prec z_1$, $\ldots, z_{n-1} \prec z_n, z_n \prec z_{n+1}, \ldots, z_{n+l-1} \prec z_{n+l} \equiv z$ are in $\mathit{\Gamma}^*$ for $l > 0$ and z_n marks the same formulas as z; In particular $z_n : \mathbf{T}C$ is in $\mathit{\Gamma}^*$ (resp. $\mathit{\Delta}^*$). Since z_n is $\prec$-saturated, $z_{n+1} : C$ is in $\mathit{\Gamma}^*$ (resp. $\mathit{\Delta}^*$). By construction $[\![z]\!] = s_{n+l}$, so $[\![z]\!] \prec^{\mathcal{K}} s_{n+l+1}$ and, by construction and inductive hypothesis, $s_{n+l+1} \Vdash C$ (resp. $s_{n+l+1} \nVdash C$). Therefore $[\![z_{n+l}]\!] \Vdash \mathbf{T}C$ (resp. $[\![z_{n+l}]\!] \nVdash \mathbf{T}C$).

If $B \equiv \mathbf{G}C$ and $z : \mathbf{G}C$ is in $\mathit{\Gamma}^*$, then, since z is $\prec$-saturated, both $z : \mathbf{T}C$ and $z : \mathbf{TG}C$ are in $\mathit{\Gamma}^*$, and, if the label $z \prec z'$ is in $\mathit{\Gamma}^*$, both $z' : C$ and $z' : \mathbf{G}C$ are in $\mathit{\Gamma}^*$. Therefore, by repeating this argument, we have that for every z'', if $z \prec z_i, \ldots, z_{i+j-1} \prec z_{i+j} \equiv z''$ are in $\mathit{\Gamma}^*$ for some $i, j \geq 0$, then $z'' : C$ and $z'' : \mathbf{G}C$ are in $\mathit{\Gamma}^*$. Note that, if z is the future looping label or z'' is inside a future loop $z_m \prec z_{m+1}, \ldots, z_{n-1} \prec z_n$ (with $n > m$) both $z_k : C$ and $z_k : \mathbf{G}C$ are in $\mathit{\Gamma}^*$ for every $m \leq k \leq n$. By inductive hypothesis, for every s, if $[\![z]\!] <^{\mathcal{K}} s$ then $s \Vdash C$, therefore $[\![z]\!] \Vdash \mathbf{G}C$.

If $z : \mathbf{G}C$ is in $\mathit{\Delta}^*$ then, by Definitions 18 and 19, we have two cases: (i) There exists some future witness label z' such that $z' : C$ is in $\mathit{\Delta}^*$ and the atoms $z \prec z_i, \ldots, z_{i+j-1} \prec z_{i+j} \equiv z'$ are in $\mathit{\Gamma}^*$ for some $i, j \geq 0$. So, by construction and inductive hypothesis there is some $s = [\![z']\!]$ such that $[\![z]\!] <^{\mathcal{K}} s$ and $s \nVdash C$, so $[\![z]\!] \nVdash \mathbf{G}C$. (ii) z is inside a future loop $z_n \prec z_{n+1}, \ldots, z_{n+i-1} \prec z_{n+i} \equiv z$, $z_{n+i} \prec z_{n+i+1}, \ldots, z_{n+l-1} \prec z_{n+l}$ (with $l \geq i$). Then there exists some label z' such that either $z_n \equiv z'$ or the atoms $z_n \prec z_{n+1}, \ldots, z_{n+q-1} \prec z_{n+q} \equiv z'$ are in $\mathit{\Gamma}^*$ for $0 \leq q \leq i$ and the formula $z' : C$ is in $\mathit{\Delta}^*$. By construction $[\![z']\!] = s_{n+q}$, so $[\![z]\!] <^{\mathcal{K}} s_{n+l+q}$ and, by inductive hypothesis, $s_{n+l+q} \nVdash C$. Therefore $[\![z]\!] \nVdash \mathbf{G}C$.

By the following result, every countermodel for the fulfilling sequent $\mathit{\Gamma}^* \Rightarrow \mathit{\Delta}^*$ is a countermodel for the corresponding endsequent $\mathit{\Gamma} \Rightarrow \mathit{\Delta}$:

Lemma 29. *All the rules of* G3LT$_{cl}$ *preserve countermodels, that is, a countermodel for (at least one of) the premisses is a countermodel for the conclusion.*

Proof. Immediate for the rules for **T** and **Y** and for the rules of seriality. For the propositional rules, by definition of validity for the propositional connectives. For the rules for **G**, **F**, **H** and **P**, by their fixed-point interpretation.

Theorem 30. *If the purely logical sequent $\mathit{\Gamma} \Rightarrow \mathit{\Delta}$ has no countermodels, then it is provable in* G3LT$_{cl}$.

Corollary 31. *Provability of purely logical sequents in* G3LT$_{cl}$ *is closed with respect to cut.*

Proof. By soundness of the cut rule and completeness of G3LT$_{cl}$.

6 Termination of Proof Search

In root-first application of the rules of G3LT$_{cl}$, two possibilities arise: (i) The proof search terminates because we find a fulfilling sequent or because every branch leads to an initial sequent or an instance of $L\bot$; (ii) The proof search does not terminate and, by König's Lemma, there is at least one infinite branch.

However, we can truncate a potentially infinite proof search as shown below. By Theorem 22, if $\Gamma \Rightarrow \Delta$ is not provable, then the proof search leads to a reduced fulfilling sequent. Whenever a branch leads to a sequent with a roundabout chain, we can drop that branch and start a new one: If every branch in the proof search for $\Gamma \Rightarrow \Delta$ leads to either an initial sequent or a sequent with a roundabout chain, then $\Gamma \Rightarrow \Delta$ is provable in G3LT$_{cl}$.

Lemma 32. *Suppose that the proof search for a purely logical sequent $\Gamma \Rightarrow \Delta$, with all the formulas labelled by x, leads to a sequent $\Gamma' \Rightarrow \Delta'$: If the chain $y_{-m} \prec y_{-(m-1)}, \ldots, y_{-1} \prec y_0 \equiv x$ and the chain $x \equiv y_0 \prec y_1, \ldots, y_{n-1} \prec y_n$ are not roundabout then the number of labels has an exponential bound on the order of the length of $A \equiv \wedge\Gamma^x \supset \vee\Delta^x$, namely $m, n \leq \sum_{i=1}^{2^{3|A|}} i$.*

Proof. (*Sketch*) We recall here that the rules of G3LT$_{cl}$ reflect the closure algorithm that from a formula A gives the set of its closure formulas and, by Corollary 9, the number of subsets of closure formulas of A is at most $2^{3|A|}$. Let us consider the longest case of a non-roundabout chain $y_0 \prec y_1, \ldots, y_{n-1} \prec y_n$ such that for every k with $0 \leq k \leq n$, y_k labels a subset of closure formulas of A. It contains a first subchain $y_0 \prec y_1, \ldots, y_{i-2} \prec y_{i-1}$ such that $i = 2^{3|A|}$ and every subset of closure formulas of A is labelled by some y_k, for $0 \leq k \leq i-1$. Then we have a second subchain $y_i \prec y_{i+1}, \ldots, y_{i+j-2} \prec y_{i+j-1}$, such that $j = 2^{3|A|}-1$ and every subset of closure formulas of A except one is marked by y_k for $i \leq k \leq i+j-1$. Thus, the subchain in the $l+1$st position contains $j = 2^{3|A|} - l$ labels, that mark the same subsets of *cl(A)* marked by the members of the chain in the lth position, except one. Summing up the numbers of the members of each subchain, we finally obtain that $n = \sum_{i=1}^{2^{3|A|}} i$. The same argument applies to the chain $y_{-m} \prec y_{-(m-1)}, \ldots, y_{-1} \prec y_0$, therefore $m = \sum_{i=1}^{2^{3|A|}} i$.

Theorem 33. *Proof search for* G3LT$_{cl}$ *terminates.*

Proof. Let us suppose that the proof search for the purely logical sequent $\Gamma \Rightarrow \Delta$ (with all its formulas labelled by x) does not terminate. Since every rule of G3LT$_{cl}$ has a finite number of premisses, any derivation tree is finitely branching, so by König's Lemma there is at least one infinite branch. Obviously it cannot lead to an initial sequent, nor to a conclusion of L$\bot$, nor to a fulfilling sequent, because otherwise it would be finite. We have to show that it contains a sequent with a roundabout chain. Note that the endsequent contains a finite number of formulas: The logical rules for connectives and for temporal operators can introduce only a finite number of new formulas, and by Lemma 6 temporal rules cannot be applied more than once with the same principal formula(s).

Furthermore, by Lemmas 3 and 4 we need not apply a seriality rule with side label z, if z is not a label in the sequent or the antecedent already contains an atom $z \prec z'$ (resp. $z' \prec z$). Consequently, an infinite branch should contain a sequent with an infinite $\prec$-chain. However, by Lemma 32 if a chain is not roundabout, then it is finite and exponentially bounded on the order of the length of the formula corresponding to the endsequent $\Gamma \Rightarrow \Delta$. Therefore, any potentially infinite branch can be truncated as soon as a sequent is met that contains a chain $z_{-m} \prec z_{-(m-1)}, \ldots, z_{-1} \prec z_0 \equiv x, z_0 \prec z_1, \ldots, z_{n-1} \prec z_n$ with $m > \sum_{i=1}^{2^{3|\wedge \Gamma^x \supset \vee \Delta^x|}} i$ or $n > \sum_{i=1}^{2^{3|\wedge \Gamma^x \supset \vee \Delta^x|}} i$.

References

1. Boretti, B.: Proof Analysis in Temporal Logic, Ph.D. Thesis, Univ. of Milan (2008)
2. Boretti, B., Negri, S.: On the finitization of Priorean linear time. In: SILFS 2007. Proceedings of the International Conference of the Italian Society for Logic and Philosophy of Science. College Publications (in press, 2009)
3. Brotherston, J., Simpson, A.: Complete sequent calculi for induction and infinite descent. In: LICS 2007. Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science, pp. 51–62. IEEE Computer Society, Washington (2007)
4. Brünnler, K., Lange, M.: Cut-free sequent systems for temporal logic. Journal of Logic and Algebraic Programming 76, 216–225 (2008)
5. Coquand, T.: Decidability Proof of LTL (unpublished note, 2007), http://www.cs.chalmers.se/~coquand/LOGIC/ltl.pdf
6. Kesten, Y., Manna, Z., McGuire, H., Pnueli, A.: Decision algorithm for full propositional temporal logic. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 97–109. Springer, Heidelberg (1993)
7. Lichtenstein, O., Pnueli, A.: Propositional temporal logics: decidability and completeness. Logic Journal of IGPL 8, 55–85 (2000)
8. Negri, S.: Proof analysis in modal logic. J. of Phil. Logic 34, 507–544 (2005)
9. Negri, S.: Proof analysis in non-classical logics. In: Dimitracopoulos, C., Newelski, L., Normann, D., Steel, J. (eds.) Logic Colloquium 2005, ASL Lecture Notes in Logic, vol. 28, pp. 107–128. Cambridge University Press, Cambridge (2007)
10. Negri, S.: Kripke completeness revisited. In: Primiero, G., Rahman, S. (eds.) Acts of Knowledge - History, Philosophy and Logic. College Publications (in press, 2009)
11. Schmitt, P.H., Goubault-Larrecq, J.: A tableau system for linear-TIME temporal logic. In: Brinksma, E. (ed.) TACAS 1997. LNCS, vol. 1217, pp. 130–144. Springer, Heidelberg (1997)
12. Schwendimann, S.: A New One-Pass Tableau Calculus for PLTL. In: de Swart, H. (ed.) TABLEAUX 1998. LNCS(LNAI), vol. 1397, pp. 277–291. Springer, Heidelberg (1998)
13. Sistla, A.P., Clarke, E.M.: The Complexity of Propositional Linear Temporal Logics. Journal of the ACM 32, 733–749 (1985)
14. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (1994)
15. Wolper, P.: The tableau method for temporal logic: An overview. Logique et Analyse 110-111, 119–136 (1985)

A Tableau-Based System for Spatial Reasoning about Directional Relations

Davide Bresolin[1,⋆], Angelo Montanari[2], Pietro Sala[2], and Guido Sciavicco[3,⋆⋆]

[1] Department of Computer Science, University of Verona, Verona, Italy
[2] Department of Mathematics and Computer Science,
University of Udine, Udine, Italy
[3] Department of Information, Engineering and Communications,
University of Murcia, Murcia, Spain

Abstract. The management of qualitative spatial information is an important research area in computer science and AI. Modal logic provides a natural framework for the formalization and implementation of qualitative spatial reasoning. Unfortunately, when directional relations are considered, modal logic systems for spatial reasoning usually turn out to be undecidable (often even not recursively enumerable). In this paper, we give a first example of a decidable modal logic for spatial reasoning with directional relations, called Weak Spatial Propositional Neighborhood Logic (WSpPNL for short). WSpPNL features two modalities, respectively an east modality and a north modality, to deal with non-empty rectangles over $\mathbb{N} \times \mathbb{N}$. We first show the NEXPTIME-completeness of WSpPNL, then we develop an optimal tableau method for it.

1 Introduction

The main goal of qualitative spatial representation and reasoning techniques is to capture common-sense knowledge about space and to provide a calculus of spatial information without referring to a quantitative model. Even though quantitative models provide a more accurate description of spatial domains, qualitative models are often the best or the only choice. In many cases, indeed, there is a lack of quantitative models or existing ones turn out to be intractable. In addition, qualitative models make it possible to cope with spatial data indeterminacy and to reason about incomplete spatial knowledge. The problem of representing and reasoning about qualitative spatial information can be viewed under three different points of view: (i) the *algebraic* perspective, that is, purely existential theories formulated as constraint satisfaction systems over jointly exhaustive and mutually disjoint sets of topological, directional, or combined relations; (ii) the *first-order* perspective, that is, first-order theories of topological, directional, or

[⋆] Davide Bresolin has been partially supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant no. INFSO-ICT-223844.

[⋆⋆] Guido Sciavicco has been partially supported by the Spanish projects PET2006-0406 and TIN2006-15460-C04-01.

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 123–137, 2009.

combined relations; (iii) the *modal logic* perspective, where a propositional modal language is interpreted over (a representation of space via) suitable Kripke structures. The increase in expressiveness of the two latter approaches is paired by an increase in computational complexity, which often makes them impracticable. Depending on the considered class of spatial relations, we can further distinguish between *topological* and *directional* spatial reasoning. While topological relations between pairs of spatial objects (viewed as sets of points) are preserved under translation, scaling, and *rotation*, directional relations depend on the relative spatial position of the objects.

A comprehensive and sufficiently up-to-date survey, which covers topological, directional, and combined constraint systems and relations, can be found in [9]. Deductive systems for reasoning about topological relations have been proposed in various papers, including Bennett's work [4,5], later extended by Bennett et al. [6], Nutt's systems for generalized topological relations [17], the modal logic systems for a number of mathematical theories of space described in [1], the logic of connectedness constraints developed by Kontchakov et al. [12], and Lutz and Wolter's modal logic of topological relations [13]. Directional relations have been mainly dealt with following both the algebraic approach and the modal logic one. As for the first one, the most important contributions are those by Güsgen [11] and by Mukerjee and Joe [16], that introduce Rectangle Algebra (RA), later extended by Balbiani et al. in [2,3]. As for the second one, we mention Venema's Compass Logic [18], whose undecidability has been shown by Marx and Reynolds in [14], and Spatial Propositional Neighborhood Logic (SpPNL for short) by Morales et al. [15], that generalizes the logic of temporal neighborhood [10] to the two-dimensional space. SpPNL makes it possible to reason about regions, approximated by their minimum bounding boxes, by taking advantage of four modal operators that allow one to move along the x- and the y-axis. In [15], the authors analyze the expressive power of the logic, provide a representation theorem, and devise a (non-terminating) sound and complete tableau system.

In this paper, we focus our attention on a proper syntactical and semantical fragment of SpPNL, called Weak SpPNL (WSpPNL for short). SpPNL has been proved to be undecidable over most relevant class of frames [15]. To recover decidability, we restrict ourselves to the class of frames isomorphic to $\mathbb{D} \times \mathbb{D}$, where $\mathbb{D}$ is either $\mathbb{N}$ or a prefix of it, and consider a syntactic fragment of SpPNL with two modalities only, namely $\langle \mathrm{E} \rangle$ (east) and $\langle \mathrm{N} \rangle$ (north), with a weakened semantics. We show that WSpPNL is NEXPTIME-complete, and we provide it with a sound and complete tableau system. Both the decidability proof and the tableau system can be viewed as non-trivial adaptations of those for Right Propositional Neighborhood Logic (RPNL) [7]. We also show that WSpPNL is expressive enough to support a (weak form of) universal operator and nominals. At the best of our knowledge, WSpPNL is the first example of a decidable modal logic for directional reasoning that deals with extended regions. For the lack of space, proofs have been omitted (they are reported in the long version of the paper that the interested reader can obtain from the authors).

2 SpPNL and WSpPNL

The language of Spatial Propositional Neighborhood Logic (SpPNL) consists of a set of propositional variables $\mathcal{AP}$, the logical connectives $\neg$ and $\vee$, and the modalities $\langle \mathrm{E} \rangle, \langle \mathrm{W} \rangle, \langle \mathrm{N} \rangle$, and $\langle \mathrm{S} \rangle$. The other logical connectives, as well as the logical constants $\top$ and $\bot$, can be defined in the usual way. Let $p \in \mathcal{AP}$. SpPNL formulas, denoted by $\varphi, \psi, \ldots$, are recursively defined as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \psi \mid \langle \mathrm{E} \rangle\varphi \mid \langle \mathrm{W} \rangle\varphi \mid \langle \mathrm{N} \rangle\varphi \mid \langle \mathrm{S} \rangle\varphi.$$

Let $\mathbb{D}_h = \langle D_h, < \rangle$ and $\mathbb{D}_v = \langle D_v, < \rangle$, where D_h (resp, D_v) is (a prefix of) the set of natural numbers $\mathbb{N}$ and $<$ is the usual linear order. Elements of $\mathbb{D}_h$ (resp., $\mathbb{D}_v$) will be denoted by $h_a, h_b, \ldots$ (resp., $v_a, v_b, \ldots$). A *spatial frame* is a structure $\mathbb{F} = \mathbb{D}_h \times \mathbb{D}_v$. The set of *objects* (rectangles) is the set $\mathbb{O}(\mathbb{F}) = \{\langle (h_a, v_b), (h_c, v_d) \rangle \mid h_a < h_c,\ v_b < v_d,\ h_a, h_c \in D_h,\ v_b, v_d \in D_v\}$. The semantics of SpPNL over $\mathbb{O}(\mathbb{F})$ is given in terms of *spatial models* $M = \langle \mathbb{F}, \mathbb{O}(\mathbb{F}), \mathcal{V} \rangle$, where $\mathbb{F}$ is a spatial frame, $\mathbb{O}(\mathbb{F})$ is the set of relevant objects, and $\mathcal{V} : \mathbb{O}(\mathbb{F}) \mapsto 2^{\mathcal{AP}}$ is a *spatial valuation function.* The pair $(\mathbb{F}, \mathbb{O}(\mathbb{F}))$ is called *spatial structure.* Given a model M and an object $\langle (h_a, v_b), (h_c, v_d) \rangle$, the *truth* relation for SpPNL formulas is defined as follows:

- $M, \langle (h_a, v_b), (h_c, v_d) \rangle \Vdash p$ iff $p \in \mathcal{V}(\langle (h_a, v_b), (h_c, v_d) \rangle)$, for any $p \in \mathcal{AP}$;
- $M, \langle (h_a, v_b), (h_c, v_d) \rangle \Vdash \neg\phi$ iff $M, \langle (h_a, v_b),\ (h_c, v_d) \rangle \nVdash \phi$;
- $M, \langle (h_a, v_b), (h_c, v_d) \rangle \Vdash \phi \vee \psi$ iff $M, \langle (h_a, v_b),\ (h_c, v_d) \rangle \Vdash \phi$ or $M,\ \langle (h_a, v_b), (h_c, v_d) \rangle \Vdash \psi$;
- $M, \langle (h_a, v_b), (h_c, v_d) \rangle \Vdash \langle \mathrm{E} \rangle\psi$ iff there exists $h_e \in D_h$ such that $h_c < h_e$, and $M, \langle (h_c, v_b), (h_e, v_d) \rangle \Vdash \psi$;
- $M\langle (h_a, v_b), (h_c, v_d) \rangle \Vdash \langle \mathrm{W} \rangle\psi$ iff there exists $h_e \in D_h$ such that $h_e < h_a$, and $M, \langle (h_e, v_b), (h_a, v_d) \rangle \Vdash \psi$;
- $M, \langle (h_a, v_b), (h_c, v_d) \rangle \Vdash \langle \mathrm{N} \rangle\psi$ iff there exists $v_e \in D_v$ such that $v_d < v_e$, and $M, \langle (h_a, v_d), (h_c, v_e) \rangle \Vdash \psi$;
- $M, \langle (h_a, v_b), (h_c, v_d) \rangle \Vdash \langle \mathrm{S} \rangle\psi$ iff there exists $v_e \in D_v$ such that $v_e < v_b$, and $M, \langle (h_a, v_e), (h_c, v_b) \rangle \Vdash \psi$.

As an example, the semantics of $\langle \mathrm{E} \rangle$ (resp., $\langle \mathrm{N} \rangle$) is graphically depicted in Figure 1 (left): if $\langle (h_a, v_b), (h_c, v_d) \rangle$ satisfies $\langle \mathrm{E} \rangle p$ (resp., $\langle \mathrm{N} \rangle p$), then there exists a rectangle whose left (resp., bottom) edge coincides with the right (resp., top) edge of $\langle (h_a, v_b),\ (h_c, v_d) \rangle$ that satisfies p.

Both the strength (expressiveness) and the weakness (undecidability) of the logic SpPNL originate from the fact that its operators allow one to move (in one step) from one rectangle to a right (resp., left, top, bottom) adjacent one. As an example, when we apply the operator $\langle \mathrm{E} \rangle$ to move to the right of the current rectangle, three out of four coordinates of the resulting rectangle, namely, h_c, v_b, v_d, are determined by (coincide with) those of the current one. The computational behavior of the logic can be improved by relaxing such a constraint. Let us define the east (resp., west, north, south) of a rectangle as the entire area to the right of it (resp., to the left of it, over it, under it) and redefine the semantics of the modal operators accordingly. The revised semantics of $\langle \mathrm{E} \rangle$ (resp.,

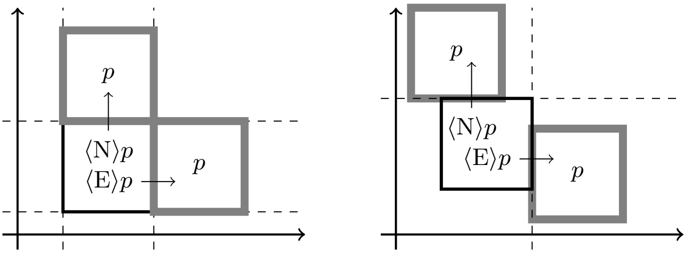


Fig. 1. The semantics of ⟨E⟩ and ⟨N⟩ in SpPNL (left) and WSpPNL (right)

⟨N⟩) is graphically depicted in Figure 1 (right). According to it, only one out of four coordinates of the resulting rectangle, namely, h_c (resp., v_d), is determined by (coincide with) those of the current one.

Weak SpPNL (WSpPNL for short) features the east ⟨E⟩ and north ⟨N⟩ modalities only, endowed with the above-described weakened semantics. The language of WSpPNL consists of a set of propositional variables $\mathcal{AP}$, the logical connectives $\neg$ and $\vee$, and the modalities ⟨E⟩ and ⟨N⟩. The other logical connectives and the logical constants $\top$ and $\bot$ are defined in the usual way. WSpPNL *formulas* are recursively defined as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \psi \mid \langle \mathrm{E} \rangle \varphi \mid \langle \mathrm{N} \rangle \varphi.$$

Given a model M and an object $\langle (h_a, v_b), (h_c, v_d) \rangle$, the clauses for the two modal operators are revised as follows:

- $M, \langle (h_a, v_b), (h_c, v_d) \rangle \Vdash \langle \mathrm{E} \rangle \psi$ iff there exist $h_e \in D_h$ and $v_f, v_g \in D_v$ such that $h_c < h_e$, $v_f < v_g$, and $M, \langle (h_c, v_f), (h_e, v_g) \rangle \Vdash \psi$;
- $M, \langle (h_a, v_b), (h_c, v_d) \rangle \Vdash \langle \mathrm{N} \rangle \psi$ iff there exist $v_e \in D_v$ and $h_f, h_g \in D_h$ such that $v_d < v_e$, $h_f < h_g$, and $M, \langle (h_f, v_d), (h_g, v_e) \rangle \Vdash \psi$.

Let [E] and [N] be the duals of ⟨E⟩ and ⟨N⟩, respectively. We say that φ is a *horizontal formula* if $\varphi = \langle \mathrm{E} \rangle \psi$ or $\varphi = [\mathrm{E}]\psi$ for some ψ (notice that $\neg\langle \mathrm{E} \rangle \psi$ is equivalent to $[\mathrm{E}]\neg\psi$ and $\neg[\mathrm{E}]\psi$ is equivalent to $\langle \mathrm{E} \rangle \neg\psi$); similarly, we say that φ is a *vertical formula* if $\varphi = \langle \mathrm{N} \rangle \psi$ or $\varphi = [\mathrm{N}]\psi$ for some ψ. *Spatial formulas* are horizontal and vertical formulas.

Since WSpPNL features only north/east operators, we can restrict our attention to the *initial object* $\langle (0,0), (1,1) \rangle$, as stated by the following proposition.

Proposition 1. *Let φ be a WSpPNL-formula. Then, φ is satisfiable if and only if the WSpPNL-formula $\varphi' = \varphi \vee \langle \mathrm{E} \rangle \varphi \vee \langle \mathrm{E} \rangle \langle \mathrm{E} \rangle \varphi \vee \langle \mathrm{N} \rangle \varphi \vee \langle \mathrm{N} \rangle \langle \mathrm{N} \rangle \varphi$ is satisfiable over the initial object.*

Thanks to Proposition 1, satisfiability of a WSpPNL-formula φ thus reduces to the existence of a model M such that $M, \langle (0,0),\ (1,1) \rangle \Vdash \varphi'$.

3 WSpPNL Expressiveness

In this section we show that, despite its simplicity, WSpPNL is expressive enough to capture various interesting spatial notions. First of all, it makes it possible to define a sort of *pseudo-universal* modal operator. As implicitly stated by Proposition 1, the lack of the south/west operators prevents WSpPNL from accessing objects whose left bottom corner is equal to $(0, 0)$. However, WSpPNL can access every other object of the frame.

Definition 1. *Given a WSpPNL-formula ψ, we say that ψ is true* almost everywhere *in a model M if and only if for every object $\langle(h_a, v_b),(h_c, v_d)\rangle$ such that $h_a \neq 0$ or $v_b \neq 0$, $M, \langle(h_a, v_b),(h_c, v_d)\rangle \Vdash \psi$.*

Let [WU] (*weakly universal*) be the following derived operator of WSpPNL:

$$[\mathrm{WU}]\psi ::= \psi \wedge [\mathrm{N}][\mathrm{E}]\psi \wedge [\mathrm{E}][\mathrm{N}]\psi.$$

The next proposition shows that the operator [WU] captures the notion introduced by Definition 1.

Proposition 2. *Let M be a spatial model and $\langle(h_a, v_b),(h_c, v_d)\rangle$ be one of its objects, with $h_a \neq 0$ or $v_b \neq 0$. We have that $M, \langle(h_a, v_b),(h_c, v_d)\rangle \Vdash [\mathrm{WU}]\psi$ if and only if ψ is true almost everywhere in M.*

Moreover, for any propositional letter $p \in \mathcal{AP}$, WSpPNL allows one to express a *weak* nominal $wn(p)$.

Definition 2. *Given a propositional variable $p \in \mathcal{AP}$, we say that p is true* almost only on *$\langle(h_a, v_b), (h_c, v_d)\rangle$, with $h_a \neq 0$ or $v_b \neq 0$, in a model M if and only if $M, \langle(h_a, v_b),(h_c, v_d)\rangle \Vdash p$ and, for every object $\langle(h'_a, v'_b), (h'_c, v'_d)\rangle \neq \langle(h_a, v_b),(h_c, v_d)$, with $h'_a \neq 0$ or $v'_b \neq 0$, $M, \langle(h'_a, v'_b), (h'_c, v'_d)\rangle \Vdash \neg p$.*

Given a propositional variable p, the operator $wn(p)$ (p *is a weak nominal*) can be expressed in WSpPNL by taking advantage of two special propositional variables $\bar{p}_h$ and $\bar{p}_v$ as follows:

$$wn(p) ::= wn_1(p) \wedge wn_2(p) \wedge wn_3(p),$$

where

$$wn_1(p) ::= p \wedge \langle\mathrm{E}\rangle p_h \wedge \langle\mathrm{N}\rangle p_v,$$

$$wn_2(p) ::= [\mathrm{WU}]((\langle\mathrm{E}\rangle p \rightarrow \neg\langle\mathrm{E}\rangle\langle\mathrm{E}\rangle p) \wedge (\langle\mathrm{N}\rangle p \rightarrow \neg\langle\mathrm{N}\rangle\langle\mathrm{N}\rangle p)), \text{ and}$$

$$wn_3(p) ::= [\mathrm{WU}](p \rightarrow \neg\langle\mathrm{E}\rangle\langle\mathrm{E}\rangle p_h \wedge \neg\langle\mathrm{N}\rangle\langle\mathrm{N}\rangle p_v).$$

Proposition 3. *Let M be a spatial model and $\langle(h_a, v_b),(h_c, v_d)\rangle$ be one of its objects, with $h_a \neq 0$ or $v_b \neq 0$. It holds that if $M, \langle(h_a, v_b),(h_c, v_d)\rangle \Vdash wn(p)$, then $M, \langle(h_a, v_b),(h_c, v_d)\rangle \Vdash p$ and, for any object $\langle(h'_a, v'_b),(h'_c, v'_d)\rangle \neq \langle(h_a, v_b),(h_c, v_d)\rangle$, with $h'_a \neq 0$ or $v'_b \neq 0$, $M, \langle(h'_a, v'_b),(h'_c, v'_d)\rangle \nVdash p$.*

As shown in [15], one of the possible measures of the expressive power of a directional-based spatial logic for rectangles is the comparison with Rectangle Algebra (RA) [16]. In RA, one considers a finite set of objects (rectangles) $O_1, \dots O_n$ and a set of constraints between pair of objects. Each constraint is a pair of Allen's IA relations that capture the relationships between the projections on the x- and the y-axis of the objects. As an example, $O_1(r_i, r_j)O_2$ means that r_i (resp., r_j) is the interval relation between the x-projections (resp., y-projections) of O_1 and O_2. In general, given an *algebraic constraint network*, the main problem is to establish whether the network is consistent, that is, if all constraints can be jointly satisfied. In [15], it has been shown that SpPNL is powerful enough to express and check the consistency of an RA-constraint network. In the following, we show that the same can be done in WSpPNL as well, exploiting the weakly universal operator and the weak nominals[1]. To this end, we take advantage of the technique used in [15]. Given an RA-constraint network with objects $O_1, \dots, O_n$, we introduce a propositional variable for every object and we force it to be a weak nominal. Moreover, we introduce additional nominals for every constraint of the network whenever necessary. In such a way, we are able to represent the network as a conjunction of WSpPNL formulas which is satisfiable if and only if the network is consistent. On the one hand, such an encoding of the consistency problem involves a blow-up in computational complexity: while an RA-constraint network can be checked for consistency in NP-time, the satisfiability problem for WSpPNL is, as we will see, NEXPTIME-complete. On the other hand, WSpPNL allows one to express a number of conditions, such as, for instance, arbitrary logical disjunctions, negations, and universal properties [8], that cannot be encoded in an RA-constraint network. Let us show now, as a source of exemplification, how WSpPNL can express the RA-constraint $O_1(d, b)O_2$ between two objects O_1 and O_2, that is, the x-projection (resp., y-) of O_1 is *during* (resp., *before*) the x-projection (resp., y-) of O_2. Let O_1, O_2, and $O_{(d,b)}$ be three propositional variables and let $\langle\exists\rangle\psi$ be a shorthand for $\langle \mathrm{N}\rangle\psi \vee \langle \mathrm{N}\rangle\langle \mathrm{N}\rangle\psi \vee \langle \mathrm{E}\rangle\psi \vee \langle \mathrm{E}\rangle\langle \mathrm{E}\rangle\psi$. The constraint "*there exist two objects O_1 and O_2 such that $O_1(d, b)O_2$*" can be expressed by the following WSpPNL formula:

$$\langle\exists\rangle wn(O_1) \wedge \langle\exists\rangle wn(O_2) \wedge \langle\exists\rangle wn(O_{(d,b)}) \wedge [\mathrm{WU}](O_1 \rightarrow \langle \mathrm{E}\rangle\langle \mathrm{E}\rangle O_{(d,b)}) \wedge$$

$$[\mathrm{WU}](O_2 \rightarrow \langle \mathrm{E}\rangle O_{(d,b)}) \wedge [\mathrm{WU}](\langle \mathrm{E}\rangle O_2 \rightarrow \langle \mathrm{E}\rangle\langle \mathrm{E}\rangle O_1) \wedge [\mathrm{WU}](O_1 \rightarrow \langle \mathrm{N}\rangle\langle \mathrm{N}\rangle O_2).$$

4 WSpPNL Decidability and Complexity

In this section we prove some basic results which are instrumental to the development of a sound and complete (terminating) tableau system for WSpPNL.

Let φ be an WSpPNL-formula to be checked for satisfiability and let $\mathcal{AP}$ be the set of its propositional variables.

[1] It worth pointing out that the restriction to frames based on natural numbers and the exclusion of objects with left bottom corner equal to $(0, \_)$ or $(\_, 0)$ do not change the status of the network (consistent/inconsistent).

Definition 3. *The* closure $\mathrm{CL}(\varphi)$ *of* φ *is the set of all sub-formulas of* φ *and of their negations (we identify* $\neg\neg\psi$ *with* ψ*). The set of* horizontal *(resp.,* vertical*)* spatial requests *of* φ *is the set* $\mathrm{HF}(\varphi)$ *(resp.,* $\mathrm{VF}(\varphi)$*) of all horizontal (resp., vertical) spatial formulas in* $\mathrm{CL}(\varphi)$*, that is,* $\mathrm{HF}(\varphi) = \{\langle \mathrm{E}\rangle\psi, [\mathrm{E}]\psi \in \mathrm{CL}(\varphi)\}$ *(resp.,* $\mathrm{VF}(\varphi) = \{\langle \mathrm{N}\rangle\psi, [\mathrm{N}]\psi \in \mathrm{CL}(\varphi)\}$*).*

Let $|\varphi|$ (the size of φ) be the number of symbols of φ. By induction on the structure of φ, we can easily prove the following proposition.

Proposition 4. *For every formula* φ*,* $|\mathrm{CL}(\varphi)|$ *is less than or equal to* $2 \cdot |\varphi|$*, while* $|\mathrm{HF}(\varphi)|$ *and* $|\mathrm{VF}(\varphi)|$ *are less than or equal to* $2 \cdot |\varphi| - 2$*.*

Definition 4. *A* φ-atom *is a set* $A \subseteq \mathrm{CL}(\varphi)$ *such that:*

- *for every* $\psi \in \mathrm{CL}(\varphi)$*,* $\psi \in A$ *iff* $\neg\psi \notin A$*;*
- *for every* $\psi_1 \vee \psi_2 \in \mathrm{CL}(\varphi)$*,* $\psi_1 \vee \psi_2 \in A$ *iff* $\psi_1 \in A$ *or* $\psi_2 \in A$*.*

We denote the set of all φ-atoms by A_φ. We have that $|A_\varphi| \leq 2^{|\varphi|}$. Atoms are connected by the following binary relations.

Definition 5. *Let* R^h_φ *(resp.,* R^v_φ*) be a binary relation over* A_φ *such that, for every pair of atoms* $A, A' \in A_\varphi$*,* $A\ R^h_\varphi\ A'$ *(resp.,* $A\ R^v_\varphi\ A'$*) if and only if, for every* $[\mathrm{E}]\psi \in \mathrm{CL}(\varphi)$ *(resp.,* $[\mathrm{N}]\psi \in \mathrm{CL}(\varphi)$*), if* $[\mathrm{E}]\psi \in A$ *(resp.,* $[\mathrm{N}]\psi \in A$*), then* $\psi \in A'$*.*

We now introduce a suitable labelling of spatial structures based on φ-atoms.

Definition 6. *We define a* φ-labelled spatial structure *(*LSS *for short) as a pair* $\mathbf{L} = ((\mathbb{F}, \mathbb{O}(\mathbb{F})), \mathcal{L})$*, where* $(\mathbb{F}, \mathbb{O}(\mathbb{F}))$ *is a spatial structure and* $\mathcal{L} : \mathbb{O}(\mathbb{F}) \mapsto A_\varphi$ *is a* labelling function *such that, for every pair of objects* $\langle (h_a, v_b), (h_c, v_d)\rangle$ *and* $\langle (h_c, v_e), (h_f, v_g)\rangle$*,* $\mathcal{L}(\langle (h_a, v_b), (h_c, v_d)\rangle)\ R^h_\varphi\ \mathcal{L}(\langle (h_c, v_e), (h_f, v_g)\rangle)$*, and for every pair of objects* $\langle (h_a, v_b), (h_c, v_d)\rangle$ *and* $\langle (h_e, v_d), (h_f, v_g)\rangle$*,* $\mathcal{L}(\langle (h_a, v_b), (h_c, v_d)\rangle)$ $R^v_\varphi\ \mathcal{L}(\langle (h_e, v_d), (h_f, v_g)\rangle)$*. An* LSS $\mathbf{L}$ *is said to be* horizontally *(resp.,* vertically*)* fulfilling *if and only if, for every horizontal (resp., vertical) formula of the type* $\langle \mathrm{E}\rangle\psi \in \mathrm{CL}(\varphi)$ *(resp.,* $\langle \mathrm{N}\rangle\psi \in \mathrm{CL}(\varphi)$*) and every object* $\langle (h_a, v_b), (h_c, v_d)\rangle$*, if* $\langle \mathrm{E}\rangle\psi \in \mathcal{L}(\langle (h_a, v_b), (h_c, v_d)\rangle)$ *(resp.,* $\langle \mathrm{N}\rangle\psi \in \mathcal{L}(\langle (h_a, v_b), (h_c, v_d)\rangle)$*), then there exists an object* $\langle (h_c, v_e), (h_f, v_g)\rangle$ *(resp.,* $\langle (h_e, v_d), (h_f, v_g)\rangle$*) such that* $\psi \in \mathcal{L}(\langle (h_c, v_e), (h_f, v_g)\rangle)$ *(resp.,* $\psi \in \mathcal{L}(\langle (h_e, v_d), (h_f, v_g)\rangle)$*). An* LSS $\mathbf{L}$ *is said to be* fulfilling *if and only if it is both horizontally and vertically fulfilling.*

A formula φ is satisfiable if and only if there exists a fulfilling LSS such that φ belongs to the labelling of the initial object, as stated by the following theorem.

Theorem 1. *A formula* φ *is satisfiable if and only if there exists a fulfilling* LSS $\mathbf{L} = (\mathbb{F}, \mathbb{O}(\mathbb{F}), \mathcal{L})$*, with* $\varphi \in \mathcal{L}(\langle (0, 0), (1, 1)\rangle)$*.*

The above theorem reduces the satisfiability problem for φ to the problem of finding a fulfilling LSS with the initial object labelled by φ. From now on, we say that a fulfilling LSS $\mathbf{L}$ *satisfies* φ if and only if $\varphi \in \mathcal{L}(\langle (0, 0), (1, 1)\rangle)$.

Since fulfilling LSSs satisfying φ may be arbitrarily large or even infinite, we must find a way to finitely establish their existence. In the following, we first give a bound on the size of finite fulfilling LSSs and then we show that in the infinite case we can safely restrict ourselves to infinite fulfilling LSSs with a finite bounded representation. To prove these results, we take advantage of the following two fundamental properties of LSSs: (i) the labellings of all objects that share the rightmost horizontal (resp., topmost vertical) coordinate must agree on horizontal (resp., vertical) spatial formulas, that is, for every $h_a, v_b, h_c, v_d, h_e, v_f, v_g$, $\langle \mathrm{E}\rangle\psi, [\mathrm{E}]\psi \in \mathcal{L}(\langle(h_a, v_b),(h_c, v_d)\rangle)$ if and only if $\langle \mathrm{E}\rangle\psi, [\mathrm{E}]\psi \in \mathcal{L}(\langle(h_e, v_f),(h_c, v_g)\rangle)$ (resp., for every $h_a, v_b, h_c, v_d, h_e, v_f, h_g$, $\langle \mathrm{N}\rangle\psi, [\mathrm{N}]\psi \in \mathcal{L}(\langle(h_a, v_b),(h_c, v_d)\rangle)$ if and only if $\langle \mathrm{N}\rangle\psi, [\mathrm{N}]\psi \in \mathcal{L}(\langle(h_e, v_f),(h_g, v_d)\rangle)$); (ii) $\frac{|\mathrm{HF}(\varphi)|}{2}$ different objects of the type $\langle(h_c, v_e),(h_f, v_g)\rangle$ are sufficient to fulfill the existential horizontal formulas belonging to the labelling of an object $\langle(h_a, v_b),(h_c, v_d)\rangle$ (and symmetrically for the vertical axis).

Definition 7. *Given an* LSS $\mathbf{L} = (\mathbb{F}, \mathbb{O}(\mathbb{F}), \mathcal{L})$ *and* $h_c \in \mathbb{D}_h$ *(resp.,* $v_d \in \mathbb{D}_v$*), we denote by* $\mathrm{REQ}_h(h_c)$ *(resp.,* $\mathrm{REQ}_v(v_d)$ *the set of all and only the horizontal (resp., vertical) formulas belonging to the labellings of the objects of the type* $\langle(h_a, v_b),(h_c, v_d)\rangle$*. The set* $\mathrm{REQ}_h(\varphi)$ *(resp.,* $\mathrm{REQ}_v(\varphi)$*) is the set of all possible sets of horizontal (resp., vertical) requests for the formula* φ*.*

In order to bound the size of finite LSSs that we must take into consideration when checking the satisfiability of a given formula φ, we determine the maximum number of times that any set in $\mathrm{REQ}_h(\varphi)$ (resp., $\mathrm{REQ}_v(\varphi)$) may appear in a given LSS.

Definition 8. *Given an* LSS $\mathbf{L} = (\mathbb{F}, \mathbb{O}(\mathbb{F}), \mathcal{L})$*, a set of points* $D'_h \subseteq D_h$ *(resp.,* $D'_v \subseteq D_v$*), and a set of horizontal (resp., vertical) formulas* $\mathcal{R} \subseteq \mathrm{HF}(\varphi)$ *(resp.,* $\mathrm{VF}(\varphi)$*), we say that* $\mathcal{R}$ occurs n times in D'_h *(resp.,* D'_v*) if and only if there exist exactly* n *distinct points* $h_{i_1}, \ldots, h_{i_n} \in D'_h$ *(resp.,* D'_v*) such that* $\mathrm{REQ}_h(h_{i_j})$ *(resp.,* $\mathrm{REQ}_v(h_{i_j})$*)* $= \mathcal{R}$*, for all* $1 \leq j \leq n$*.*

The main technical ingredient of the proof is given by the following lemmas that, given a fulfilling LSS, show when and how it is possible to remove a point from it in such a way that the resulting LSS is still fulfilling. From now on, let $m_h = \frac{|\mathrm{HF}(\varphi)|}{2}$ and $m_v = \frac{|\mathrm{VF}(\varphi)|}{2}$.

Lemma 1. *Let* $\mathbf{L} = (\mathbb{F}, \mathbb{O}(\mathbb{F}), \mathcal{L})$ *be a fulfilling* LSS *that satisfies* φ*. If there exists a point* $h_{i_l} \in D_h$*, with* $h_{i_l} > 0$*, such that there are at least* $m_v \cdot m_h + m_v$ *points* $0 < h_{i_j} < h_{i_l}$ *and at least* $m_h + m_v$ *points* $h_{i_l} < h_{i_j}$ *such that, for every* j*,* $\mathrm{REQ}_h(h_{i_l}) = \mathrm{REQ}_h(h_{i_j})$*, then there exists a fulfilling* LSS $\overline{\mathbf{L}} = (\overline{\mathbb{F}}, \mathbb{O}(\overline{\mathbb{F}}), \overline{\mathcal{L}})$ *that satisfies* φ*, with* $\overline{D}_h = D_h \setminus \{h_{i_l}\}$ *and* $\overline{D}_v = D_v$*.*

Lemma 1 can be intuitively explained as follows. When we remove a “horizontal” point h_{i_l}, that is, all points with horizontal coordinate equal to h_{i_l} (in fact, removing h_{i_l} means removing all objects having h_{i_l} as their leftmost or rightmost horizontal coordinate), we can introduce one or more *defects* in $\mathbf{L}$. Such defects

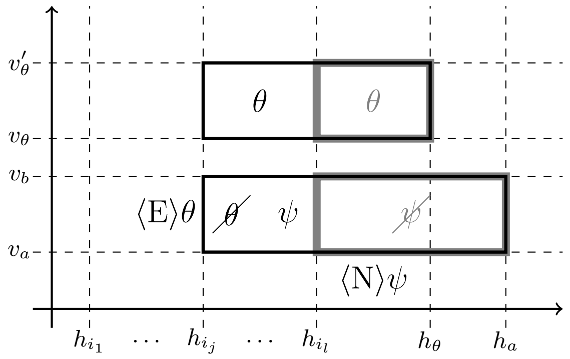


Fig. 2. Fixing defects of type 3

can be of three different types, depending on which kind of existential formulas are no more satisfied as an effect of the removal of h_{i_l}. A defect of type 1 is generated by an $\langle E\rangle\psi$ formula belonging to the set of requests of a point to the west of h_{i_l}. Such a defect can be immediately fixed by taking advantage of the copies of h_{i_l} to the east of it. A defect of type 2 is generated by an $\langle N\rangle\psi$ formula belonging to the set $\mathrm{REQ}_v(v_a)$, where v_a is the bottommost vertical coordinate of an object with rightmost horizontal coordinate h_{i_l}. As in the previous case, such a defect can be immediately fixed by using the copies of h_{i_l} to the east of it. A defect of type 3 is generated by an $\langle N\rangle\psi$ formula belonging to the set $\mathrm{REQ}_v(v_a)$, where v_a is the bottommost vertical coordinate of an object with leftmost horizontal coordinate h_{i_l}. As shown in Fig. 2, to fix a defect of this type, we take advantage of the copies of h_{i_j} to the west of it. Replacing θ by ψ in the labeling of the object $\langle(h_{i_j}, v_a), (h_a, v_b)\rangle$ may possibly introduce a new defect ($\langle E\rangle\theta$). Thanks to the availability of a sufficient number of copies of h_{i_l} to the west of it, we can guarantee that such a new defect may involve a horizontal request only and it can be solved by forcing h_{i_j} to behave as h_{i_l} behaved.

Lemma 2. *Let* $\mathbf{L} = (\mathbb{F}, \mathbb{O}(\mathbb{F}), \mathcal{L})$ *be a fulfilling* LSS *that satisfies* φ*. If there exists a point* $v_{i_l} > 0 \in D_v$ *such that there are at least* $m_v \cdot m_h + m_h$ *points* $0 < v_{i_j} < v_{i_l}$ *and at least* $m_h + m_v$ *points* $v_{i_l} < v_{i_j}$ *such that, for every* j, $\mathrm{REQ}_v(v_{i_l}) = \mathrm{REQ}_v(v_{i_j})$*, then there exists a fulfilling* LSS $\overline{\mathbf{L}} = (\overline{\mathbb{F}}, \mathbb{O}(\overline{\mathbb{F}}), \overline{\mathcal{L}})$ *that satisfies* φ*, with* $\overline{D}_h = D_h$ *and* $\overline{D}_v = D_v \setminus \{v_{i_l}\}$*.*

The above lemmas can be directly exploited to give a bound on finite LSSs.

Theorem 2. *Let* $\mathbf{L} = (\mathbb{F}, \mathbb{O}(\mathbb{F}), \mathcal{L})$ *be a finite fulfilling* LSS *that satisfies* φ*. Then, there exists a finite fulfilling* LSS $\overline{\mathbf{L}} = (\overline{\mathbb{F}}, \mathbb{O}(\overline{\mathbb{F}}), \overline{\mathcal{L}})$ *that satisfies* φ *such that, for every* $\overline{h}_i \in \overline{D}_h$ *(resp.,* $\overline{v}_j \in \overline{D}_v$*),* $\mathrm{REQ}_h(\overline{h}_i)$ *occurs at most* $m_v \cdot m_h + 2 \cdot m_v + m_h$ *times in* $\overline{D}_h \setminus \{0\}$ *(resp.,* $\mathrm{REQ}_v(\overline{v}_j)$ *occurs at most* $m_v \cdot m_h + 2 \cdot m_h + m_v$ *times in* $\overline{D}_v \setminus \{0\}$*).*

Infinite structures can be dealt with as follows. First of all, we must distinguish among three types of infinite LSSs, depending on whether only one domain is infinite (and which one) or both. For each of them, we introduce an appropriate representation.

Definition 9. *An infinite* LSS $\mathbf{L} = (\mathbb{F}, \mathbb{O}(\mathbb{F}), \mathcal{L})$ *is* horizontally ultimately periodic*, with prefix l_h and period $p_h > 0$, if and only if for all $i > l_h$,* $\mathrm{REQ}_h(h_i) = \mathrm{REQ}_h(h_{i+p_h})$*; it is* vertically ultimately periodic*, with prefix l_v and period $p_v > 0$, if and only if for all $j > l_v$,* $\mathrm{REQ}_v(v_j) = \mathrm{REQ}_v(v_{j+p_v})$*; it is simply* ultimately periodic *if it is (i) both horizontally and vertically ultimately periodic, or (ii) horizontally ultimately periodic and vertically finite, or (iii) horizontally finite and vertically ultimately periodic.*

The proof for the infinite case essentially reduces to show that for any infinite fulfilling LSS there exists an equivalent ultimately periodic fulfilling LSS whose horizontal and/or vertical prefixes and periods satisfy suitable bounds. In case of structures which are infinite in one dimension only, say, the horizontal one, the search for an ultimately periodic characterization of this component can be paired with the application of the argument of Theorem 2 to the other component, say, the vertical one (the case in which the vertical component is infinite and the horizontal one is finite is fully symmetric). Let us assume the finite vertical component to be bounded, that is, for each $\overline{v}_j \in \overline{D}_v$, $\mathrm{REQ}_v(\overline{v}_j)$ occurs at most $m_v \cdot m_h + 2 \cdot m_h + m_v$ times in $\overline{D}_v \setminus \{0\}$. The following theorem holds.

Theorem 3. *Let* $\mathbf{L} = (\mathbb{F}, \mathbb{O}(\mathbb{F}), \mathcal{L})$ *be a horizontally infinite fulfilling* LSS *that satisfies φ. Then, there exists a horizontally ultimately periodic fulfilling* LSS $\overline{\mathbf{L}} = (\overline{\mathbb{F}}, \mathbb{O}(\overline{\mathbb{F}}), \overline{\mathcal{L}})$*, with prefix l_h and period p_h, that satisfies φ such that:*

1. *for every set of requests $\mathcal{R}$ that occurs only finitely often in $\overline{\mathbf{L}}$, $\mathcal{R}$ appears at most $m_v \cdot m_h + 2 \cdot m_v + m_h$ times in the set $\{\overline{h}_j \mid j \leq l_h\}$;*
2. *for every set of requests $\mathcal{R}$ that occurs infinitely often in $\overline{\mathbf{L}}$, $\mathcal{R}$ appears at most $m_v \cdot m_h + m_v$ times in the set $\{\overline{h}_j \mid j \leq l_h\}$;*
3. *for every pair of points $\overline{h}_a, \overline{h}_b \in \overline{D}_h$, with $\overline{h}_{l_h} < \overline{h}_a, \overline{h}_b \leq \overline{h}_{l_h+p_h}$, if $a \neq b$, then* $\mathrm{REQ}_h(\overline{h}_a) \neq \mathrm{REQ}_h(\overline{h}_b)$.

By applying a similar process to the vertical component, it is possible to get an ultimately periodic counterpart to any LSS, which is infinite in the vertical dimension or in both dimensions. Hence, the search for LSS (models) satisfying a given WSpPNL-formula can be confined to the structures of Definition 9.

Taking advantage of Theorem 2 and Theorem 3, we can devise a simple decision procedure for WSpPNL, that restricts the search for a model satisfying φ to finite exponential (pseudo-)models (such a decision procedure can be viewed as a generalization of the one for RPNL given in [7]). It immediately follows that the satisfiability problem for WSpPNL is in NEXPTIME. To prove NEXPTIME-hardness, we will reduce the satisfiability problem for RPNL over natural numbers (which has been shown to be NEXPTIME-hard in [7]) to it.

RPNL is the future fragment of the interval logic of temporal neighborhood. Formulas of RPNL are built on by using propositional variables, logical connectives, and the neighborhood modality $\langle A \rangle$ according to the grammar:

$$f ::= p \mid \neg f \mid f \vee g \mid \langle A \rangle f$$

RPNL is interpreted over models of the form $M = \langle \mathbb{D}, \mathbb{I}(\mathbb{D}), \mathcal{V} \rangle$, where $\mathbb{D} = \langle D, < \rangle$ is $\mathbb{N}$ or a prefix of it, $\mathbb{I}(\mathbb{D}) = \{[d_i, d_j] \mid d_i < d_j, d_i, d_j \in D\}$ is the set of all intervals over D, and $\mathcal{V}$ is the evaluation function. The semantics of $\langle A \rangle$ is such that $M, [d_i, d_j] \Vdash \langle A \rangle f$ iff $\exists d_k \in D$, with $d_k > d_j$, such that $M, [d_j, d_k] \Vdash f$.

Let us consider now an encoding η of RPNL formulas into WSpPNL that makes no change to the original formula except for the replacement of $\langle A \rangle$ with $\langle E \rangle$. It is not difficult to prove the next lemma.

Lemma 3. *Let φ be an RPNL-formula. We have that φ is satisfiable if and only if the WSpPNL-formula $\eta(\varphi) \wedge [\mathrm{E}][\mathrm{N}]\bot$ is satisfiable.*

Hence, we have the following theorem.

Theorem 4. *The satisfiability problem for WSpPNL is NEXPTIME-complete.*

5 The Tableau Method

In this section, we present a sound and complete (terminating) tableau method for WSpPNL based on the model-theoretic results of the previous section. We assume the reader to be familiar with the standard notions of *decorated tree, node in a tree, leaf, branch*, and *height* of a tree.

Definition 10. *Given any WSpPNL-formula φ to be checked for satisfiability, a* tableau *for φ is a suitable decorated tree $\mathcal{T}_\varphi$. Each node of $\mathcal{T}_\varphi$ is labelled with a tuple of the type $\langle \psi, \langle (h_a, v_b), (h_c, v_d) \rangle, D_h, D_v \rangle$ where $\psi \in \mathrm{CL}(\varphi)$, D_h and D_v are finite linearly-ordered sets, and $\langle (h_a, v_b), (h_c, v_d) \rangle \in \mathbb{O}(\mathbb{F})$, where $\mathbb{F}$ is the spatial frame obtained from D_h and D_v.*

Given a tableau $\mathcal{T}_\varphi$ and a branch B of it, we denote with D_h^B (resp. D_v^B) the linear order D_h (resp., D_v) associated with the leaf of B. Moreover, we denote by $\Gamma_B(\langle (h_a, v_b), (h_c, v_d) \rangle)$ the set $\{\psi \mid \langle \psi, \langle (h_a, v_b), (h_c, v_d) \rangle, D_h, D_v \rangle \in B\}$. Let $N = \{n_1, ..., n_k\}$ be a finite set of nodes. We denote by $B \cdot N$ the expansion $B \cdot n_1 \mid ... \mid n_k$, obtained by adding k immediate successors to the leaf of B. Given a finite linear order $D = \{d_0, ..., d_m\}$ and a point $d \notin D$, we denote by $D \cup \{d_i < d < d_{i+1}\}$ (resp., $D \cup \{d_m < d\}$) the linear order obtained from D adding d in between d_i and d_{i+1} (resp., after d_m)[2].

Given a tableau $\mathcal{T}_\varphi$ for a WSpPNL-formula φ and one of its branches B, for every horizontal (resp., vertical) coordinate $h \in D_h^B$ (resp., $v \in D_v^B$), we define the set of its horizontal (resp., vertical) requests $\mathrm{REQ}_h^B(h)$ (resp., $\mathrm{REQ}_v^B(v)$) as the smallest set satisfying the following properties:

- if exists $n = \langle \langle \mathrm{E} \rangle \psi, \langle (h_a, v_b), (h, v_d) \rangle, D_h^B, D_v^B \rangle$ in B, then $\langle \mathrm{E} \rangle \psi \in \mathrm{REQ}_h^B(h)$;
- if exists $n = \langle \langle \mathrm{N} \rangle \psi, \langle (h_a, v_b), (h_c, v) \rangle, D_h^B, D_v^B \rangle$ in B, then $\langle \mathrm{N} \rangle \psi \in \mathrm{REQ}_v^B(v)$;
- if exists $n = \langle [\mathrm{E}] \psi, \langle (h_a, v_b), (h, v_d) \rangle, D_h^B, D_v^B \rangle$ in B, then $[\mathrm{E}] \psi \in \mathrm{REQ}_h^B(h)$;

[2] Hereafter, for the sake of simplicity, we will denote both cases as $D \cup \{d_i < d < d_{i+1}\}$ with the implicit assumption that d_{i+1} is missing whenever $d_i = d_m$.

- if exists $n = \langle [\mathrm{N}]\psi, \langle (h_a, v_b), (h_c, v)\rangle, D_h^B, D_v^B\rangle$ in B, then $[\mathrm{N}]\psi \in \mathrm{REQ}_v^B(v)$;
- if exists $n = \langle \psi, \langle (h, v_b), (h_c, v_d)\rangle, D_h^B, D_v^B\rangle$ in B and $\langle \mathrm{E}\rangle\psi \in \mathrm{CL}(\varphi)$, then $\langle \mathrm{E}\rangle\psi \in \mathrm{REQ}_h^B(h)$;
- if exists $n = \langle \psi, \langle (h_a, v), (h_c, v_d)\rangle, D_h^B, D_v^B\rangle$in B and $\langle \mathrm{N}\rangle\psi \in \mathrm{CL}(\varphi)$, then $\langle \mathrm{N}\rangle\psi \in \mathrm{REQ}_v^B(v)$.

Rules. Let $\mathcal{T}_\varphi$ be a tableau for a WSpPNL-formula φ and let B be a branch of it. The following rules can be applied to B:

- **Not-rule:** if there exists a node labelled with $\langle \neg\neg\psi, \langle (h_a, v_b), (h_c, v_d)\rangle, D_h, D_v\rangle$ and $\psi \notin \Gamma_B(\langle (h_a, v_b), (h_c, v_d)\rangle)$, then expand B to $B \cdot n$, where $n = \langle \psi, \langle (h_a, v_b), (h_c, v_d)\rangle, D_h^B, D_v^B\rangle$;
- **And-rule:** if there exists a node labelled with $\langle \neg(\psi_1 \vee \psi_2), \langle (h_a, v_b), (h_c, v_d)\rangle, D_h, D_v\rangle \in B$ and $\{\neg\psi_1, \neg\psi_2\} \not\subseteq \Gamma_B(\langle (h_a, v_b), (h_c, v_d)\rangle)$, then expand B to $B \cdot n_1 \cdot n_2$, where $n_1 = \langle \neg\psi_1, \langle (h_a, v_b), (h_c, v_d)\rangle, D_h^B, D_v^B\rangle$ and $n_2 = \langle \neg\psi_2, \langle (h_a, v_b), (h_c, v_d)\rangle, D_h^B, D_v^B\rangle$ (if one between $\neg\psi_1$ and $\neg\psi_2$ already belongs to $\Gamma_B(\langle (h_a, v_b), (h_c, v_d)\rangle)$, we can avoid to add the corresponding node);
- **Or-rule:** if exists $\langle (\psi_1 \vee \psi_2), \langle (h_a, v_b), (h_c, v_d)\rangle, D_h, D_v\rangle \in B$, and $\{\psi_1, \psi_2\} \cap \Gamma_B(\langle (h_a, v_b), (h_c, v_d)\rangle) = \emptyset$, then expand B to $B \cdot n_1 | n_2$, where $n_1 = \langle \psi_1, \langle (h_a, v_b), (h_c, v_d)\rangle, D_h^B, D_v^B\rangle$ and $n_2 = \langle \psi_2, \langle (h_a, v_b), (h_c, v_d)\rangle, D_h^B, D_v^B\rangle$;
- **DiamondE-rule:** if for some point $h_a \in D_h^B$ it holds that $\langle \mathrm{E}\rangle\psi \in \mathrm{REQ}_h^B(h_a)$ and, for every $h_c \in D_h^B$, with $h_c > h_a$, and every $v_b, v_d \in D_v^B$, $\psi \notin \Gamma_B(\langle (h_a, v_b), (h_c, v_d)\rangle)$, then we expand B as follows. Let $\overline{h}, \overline{v}, \overline{v}'$ three fresh points. We define the following classes of nodes:
  - $n_l^{(i,j)} = \langle \psi, \langle (h_a, v_l), (h_i, v_j)\rangle, D_h, D_v\rangle$;
  - $m_l^{(i,j)} = \langle \psi, \langle (h_a, \overline{v}'), (h_i, v_j)\rangle, D_h, D_v \cup \{v_l < \overline{v}' < v_{l+1}\}\rangle$;
  - $\overline{n}_l^{(i,j)} = \langle \psi, \langle (h_a, v_l), (\overline{h}, v_j)\rangle, D_h \cup \{h_i < \overline{h} < h_{i+1}\}, D_v\rangle$;
  - $\overline{m}_l^{(i,j)} = \langle \psi, \langle (h_a, \overline{v}'), (\overline{h}, v_j)\rangle, D_h \cup \{h_i < \overline{h} < h_{i+1}\}, D_v \cup \{v_l < \overline{v}' < v_{l+1}\}\rangle$;
  - $\widehat{n}_l^{(i,j)} = \langle \psi, \langle (h_a, v_l), (h_i, \overline{v})\rangle, D_h, D_v \cup \{v_j < \overline{v} < v_{j+1}\}\rangle$;
  - $\widehat{m}_l^{(i,j)} = \langle \psi, \langle (h_a, \overline{v}'), (h_i, \overline{v})\rangle, D_h, D_v \cup \{v_l < \overline{v}' < v_{l+1}, v_j < \overline{v} < v_{j+1}\}\rangle$;
  - $\widetilde{n}_l^{(i,j)} = \langle \psi, \langle (h_a, v_l), (\overline{h}, \overline{v})\rangle, D_h \cup \{h_i < \overline{h} < h_{i+1}\}, D_v \cup \{v_j < \overline{v} < v_{j+1}\}\rangle$;
  - $\widetilde{m}_l^{(i,j)} = \langle \psi, \langle (h_a, \overline{v}'), (\overline{h}, \overline{v})\rangle, D_h \cup \{h_i < \overline{h} < h_{i+1}\}, D_v \cup \{v_l < \overline{v}' < v_{l+1}, v_j < \overline{v} < v_{j+1}\}\rangle$.

  Let $N = \{n_l^{(i,j)} | a+1 \le i \le |D_h^B| - 1 \wedge 0 \le j \wedge l \le |D_v^B| - 1 \wedge l \le j\} \cup \ldots \cup \{\widetilde{m}_l^{(i,j)} | a+1 \le i \le |D_h^B| - 1 \wedge 0 \le j \wedge l \le |D_v^B| - 1 \wedge l \le j\}$. We expand B to $B \cdot N$;
- **DiamondN-rule:** analogous to the previous case;
- **BoxE-rule:** if for some point $h_a \in D_h^B$ we have that $[\mathrm{E}]\psi \in \mathrm{REQ}_h^B(h_a)$ and there exist three points $h_c \in D_h^B$, $v_b, v_d \in D_v^B$, such that $\psi \notin \Gamma_B(\langle (h, v_b), (h_c, v_d)\rangle)$, then we expand B to $B \cdot n$ where $\langle \psi, \langle (h, v_b), (h_c, v_d)\rangle, D_h^B, D_v^B\rangle$;
- **BoxN-rule:** analogous to the previous case.

The behavior of the DiamondE-rule can be explained as follows. Suppose that we are trying to build a model for the formula φ and, at a certain stage of the construction, we find an object labeled by $\langle \mathrm{E} \rangle \psi$. We have to foresee all possible ways of satisfying the request $\langle \mathrm{E} \rangle \psi$, namely, ψ can be satisfied on an object that has been already introduced in the model (node class $n_l^{(i,j)}$) or on a new one. In the latter case, the new object can be created by adding at most one point in the horizontal component and at most two points in the vertical one, in all possible ways with respect to the existing points. This forces us to consider seven distinct classes of new nodes.

Expansion Strategy. We introduce the notions of fulfilled branch, closed (and open) branch, and blocked (and non-blocked) branch, and we describe how the expansion rules must be applied in order to guarantee the completeness of the method. We say that $\langle \mathrm{E} \rangle \psi \in \mathrm{HF}(\varphi)$ (resp., $\langle \mathrm{N} \rangle \psi \in \mathrm{VF}(\varphi)$) is *fulfilled for h by h'* (resp., *fulfilled for v by v'*) if there exists a node $n = \langle \psi, \langle (h, v_b), (h', v_d) \rangle, D_h^B, D_v^B \rangle$ (resp., $n = \langle \psi, \langle (h_a, v), (h_c, v') \rangle, D_h^B, D_v^B \rangle$) in B.

Definition 11. *Let $\mathcal{T}_\varphi$ be a tableau for a WSpPNL-formula φ and B be one of its branches. We say that B is* horizontally *(resp.,* vertically*)* fulfilled *if there exist two points $h_p < h_q \in D_h^B$ (resp., $v_p < v_q \in D_v^B$) such that the following conditions are respected:*

1. *for every $h \leq h_q$ (resp., $v \leq v_q$), every formula $\langle \mathrm{E} \rangle \psi \in \mathrm{REQ}_h^B(h)$ (resp., $\langle \mathrm{N} \rangle \psi \in \mathrm{REQ}_v^B(v)$) is fulfilled in B;*
2. *for every point $h' \geq h_p$ (resp., $v' \geq v_p$), there exists a point $h'' < h_p$ (resp., $v'' < v_p$) such that $\mathrm{REQ}_h^B(h') = \mathrm{REQ}_h^B(h'')$ (resp., $\mathrm{REQ}_v^B(v') = \mathrm{REQ}_v^B(v'')$);*
3. *for every point $h' \geq h_q$ (resp., $v' \geq v_q$), there exists a point $h_p \leq h'' \leq h_q$ (resp., $v_p \leq v'' \leq v_q$) such that $\mathrm{REQ}_h^B(h') = \mathrm{REQ}_h^B(h'')$ (resp., $\mathrm{REQ}_v^B(v') = \mathrm{REQ}_v^B(v'')$).*

The notion of horizontally (resp., vertically) fulfilled branch can be explained as follows. If each existential formula in B is explicitly fulfilled, we can choose the greatest element of D_h^B (resp., D_v^B) as h_q (resp., v_q) and any other (distinct) element of D_h^B (resp., D_v^B) as h_p (resp., v_p). This deals with the case of finite models. If there exist some existential formulas in B which are not explicitly fulfilled, it is possible to show that the satisfaction of the conditions of Definition 11 guarantees the existence of an infinite model for φ (in fact, it allows us to produce a finite representation of an ultimately periodic model for φ).

Definition 12. *Let $\mathcal{T}_\varphi$ be a tableau for a WSpPNL-formula φ and let B be one of its branches. We say that B is* closed *if and only if there exist four points $h_a, h_c \in D_h^B$ and $v_b, v_d \in D_v^B$ such that $\{\psi, \neg\psi\} \subseteq \Gamma_B(\langle (h_a, v_b), (h_c, v_d) \rangle)$; otherwise, we say that it is* open.

Definition 13. *Let $\mathcal{T}_\varphi$ be a tableau for a WSpPNL-formula φ and let B be one of its branches. We say that B is* blocked *if and only if one of the following conditions hold:*

1. *there exists a point* $h \in D_h^B$ *such that* $\mathrm{REQ}_h^B(h)$ *occurs* $m_v \cdot m_h + 2 \cdot m_v + m_h + 1$ *times in* D_h^B*;*
2. *there exists a point* $v \in D_v^B$ *such that* $\mathrm{REQ}_v^B(v)$ *occurs* $m_v \cdot m_h + 2 \cdot m_h + m_v + 1$ *times in* D_v^B*.*

Given a WSpPNL-formula φ, the *initial tableau* $\mathcal{T}_\varphi$ for φ is a single-node tree labelled by $\langle \varphi, \langle (h_0, v_0), (h_1, v_1) \rangle, \{h_0 < h_1\}, \{v_0 < v_1\} \rangle$. We expand the tableau by applying to its open branches B the following rules (in the given order):

1. apply the **Not/And/Or**-rules until they generate no new nodes in $\mathcal{T}_\varphi$;
2. apply the **BoxE/BoxN**-rules until they generate no new nodes in $\mathcal{T}_\varphi$;
3. if B is not blocked and it is not horizontally (resp., vertically) fulfilled, apply the **DiamondE-rule** (resp., **DiamondN-rule**) to it.

Definition 14. *Given a WSpPNL-formula* φ *and a tableau* $\mathcal{T}_\varphi$ *for it, we say that* $\mathcal{T}_\varphi$ *is* final *if and only if the application of the expansion strategy to every open branch of* $\mathcal{T}_\varphi$ *does not generate new nodes. A final* $\mathcal{T}_\varphi$ *is said to be* open *if there exists a vertically and horizontally fulfilled open branch* B *in it.*

Soundness and Completeness. To prove that the method is sound, we take a open branch B, that is both horizontally and vertically fulfilled, and show how to obtain a model for the formula φ from it.

Theorem 5. *If* φ *is a WSpPNL-formula and* $\mathcal{T}_\varphi$ *is an open final tableau for it, then* φ *is satisfiable.*

To prove that the method is complete it suffices to show that, for each LSS satisfying either the conditions of Theorem 2 or those of Theorem 3, there exists a corresponding horizontally and vertically fulfilled open branch in the (generated) final tableau $\mathcal{T}_\varphi$ for φ.

Theorem 6. *If* φ *is a satisfiable WSpPNL-formula, then there exists a final tableau* $\mathcal{T}_\varphi$ *for it.*

6 Conclusions and Open Problems

In the paper, we introduced and studied a decidable modal logic for spatial reasoning about directional relations. In particular, we proved its NEXPTIME-completeness and we provided it with an optimal tableau-based decision procedure. The achieved results can be generalized in several directions. First, the logic can be extended to the three-dimensional case, and beyond. The restriction to two directions only can be removed as well. In both cases, the resulting logic preserves decidability. Moreover, we believe it is possible to extend it to dense domains without loosing decidability. Other extensions seem to be more problematic from the decidability point of view. We are currently thinking of the possibility of constraining adjacent rectangles to overlap with respect to one dimension or of restricting the east (resp., north) of a rectangle to the area to the north-east of it only and to redefine the semantics of the modal operators accordingly (both restrictions can be easily lifted to the case of higher-dimensional structures).

References

1. Aiello, M., van Benthem, J.: A modal walk through space. Journal of Applied Non-Classical Logic 12(3-4), 319–363 (2002)
2. Balbiani, P., Condotta, J.F., Fariñas del Cerro, L.: A model for reasoning about bidimensional temporal relations. In: Proc. of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR 1998), pp. 124–130 (1998)
3. Balbiani, P., Condotta, J.F., Fariñas del Cerro, L.: A new tractable subclass of the rectangle algebra. In: Proc. of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-1999), pp. 442–447 (1999)
4. Bennett, B.: Spatial reasoning with propositional logics. In: Doyle, J., Sandewall, E., Torasso, P. (eds.) Proc. of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR 1994), pp. 51–62. Morgan Kaufmann, San Francisco (1994)
5. Bennett, B.: Modal logics for qualitative spatial reasoning. Journal of the Interest Group in Pure and Applied Logic (IGPL) 4(1), 23–45 (1996)
6. Bennett, B., Cohn, A.G., Wolter, F., Zakharyaschev, M.: Multi-dimensional modal logic as a framework for spatio-temporal reasoning. Applied Intelligence 17(3), 239–251 (2002)
7. Bresolin, D., Montanari, A., Sciavicco, G.: An optimal decision procedure for right propositional neighborhood logic. Journal of Automated Reasoning 4(3), 305–330 (2007)
8. Chittaro, L., Montanari, A.: Temporal representation and reasoning in artificial intelligence: Issues and approaches. Annals of Mathematics and Artificial Intelligence 28(1-4), 47–106 (2000)
9. Cohn, A.G., Hazarika, S.M.: Qualitative spatial representation and reasoning: An overview. Fundamenta Informaticae 46(1-2), 1–29 (2001)
10. Goranko, V., Montanari, A., Sciavicco, G.: Propositional interval neighborhood temporal logics. Journal of Universal Computer Science 9(9), 1137–1167 (2003)
11. Güsgen, H.: Spatial reasoning based on Allen's temporal logic. Technical Report ICSI TR89-049, International Computer Science Institute (1989)
12. Kontchakov, R., Pratt-Hartmann, I., Wolter, F., Zakharyaschev, M.: On the computational complexity of spatial logics with connectedness constraints. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS, vol. 5330, pp. 574–589. Springer, Heidelberg (2008)
13. Lutz, C., Wolter, F.: Modal logics of topological relations. Logical Methods in Computer Science 2(2) (2006)
14. Marx, M., Reynolds, M.: Undecidability of compass logic. Journal of Logic and Computation 9(6), 897–914 (1999)
15. Morales, A., Navarrete, I., Sciavicco, G.: A new modal logic for reasoning about space: spatial propositional neighborhood logic. Annals of Mathematics and Artificial Intelligence 51(1), 1–25 (2007)
16. Mukerjee, A., Joe, G.: A qualitative model for space. In: Proc. of the of the Eighth National Conference on Artificial Intelligence (AAAI-1990), pp. 721–727 (1990)
17. Nutt, W.: On the translation of qualitative spatial reasoning problems into modal logics. In: Burgard, W., Christaller, T., Cremers, A.B. (eds.) KI 1999. LNCS, vol. 1701, pp. 113–124. Springer, Heidelberg (1999)
18. Venema, Y.: Expressiveness and completeness of an interval tense logic. Notre Dame Journal of Formal Logic 31(4), 529–547 (1990)

Terminating Tableaux for the Basic Fragment of Simple Type Theory

Chad E. Brown and Gert Smolka

Saarland University, Saarbrücken, Germany

Abstract. We consider the basic fragment of simple type theory, which restricts equations to base types and disallows lambda abstractions and quantifiers. We show that this fragment has the finite model property and that satisfiability can be decided with a terminating tableau system. Both results are with respect to standard models.

1 Introduction

We are interested in higher-order fragments of classical simple type theory [1,2] for which it is decidable whether a formula is satisfied by a standard model. Only few such fragments are known:

- The propositional fragment, which is obtained by admitting no other base type but the type of truth values. In this case decidability follows from the fact that all types are interpreted as finite sets.
- The fragment consisting of disequations $s \neq t$ where s and t are pure terms that do not involve the type of truth values. The decidability follows from the completeness of lambda conversion [3].
- The fragments that correspond to propositional modal logics with inductive expressivity, for instance PDL [4] and the propositional μ-calculus [5].

In this paper we will show that the fragment of simple type theory that restricts equations to base types and disallows lambda abstraction and quantification is decidable. We call the formulas of this fragment *basic*. Here are examples of unsatisfiable basic formulas:

1. $h(h\bot{=}h\neg\bot) \neq h\bot$ $\qquad h : o\iota$
2. $h(f(f(fx))) \neq h(fx)$ $\qquad x : o,\ f : oo,\ h : o\iota$
3. $x{\neq}y \wedge gx{=}y \wedge gy{=}x \wedge f(f(fx)){=}g(fx)$ $\qquad a, x, y : o,\ f, g : oo$
4. $x{\neq}y \wedge gx{=}y \wedge gy{=}x \wedge pg \wedge \neg p(\neg)$ $\qquad x, y : o,\ g : oo,\ p : (oo)o$
5. $qfx \wedge f(fx) \wedge f(qfx){\neq}fx$ $\qquad x : o,\ f : oo,\ q : (oo)oo$

None of the formulas is a formula of standard first-order logic. Seen from the perspective of first-order logic, basic formulas are quantifier-free formulas where terms can be formulas and predicates and functions can be higher-order.

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 138–151, 2009.

Most of the above formulas are out of the reach of the automated tactics of Isabelle [6] and the higher-order provers TPS [7] and LEO-II [8]. We hope that the techniques of this paper will contribute to better auto tactics for higher-order proof assistants.

Our decision procedure comes in the form of a terminating tableau system, which is a subsystem of a tableau system for full extensional type theory. The extended system is the dual of a Henkin-complete cut-free one-sided sequent calculus devised by Brown [9], which has been the starting point for the research reported in this paper. The most difficult part of the correctness proof for the terminating system is a model existence theorem, which we establish with the possible-values technique. The possible-values technique originated with cut elimination proofs [10,11] and has been used by Brown [9] to obtain Henkin models. We seem to be the first to obtain standard models with the possible-values technique.

2 Basic Definitions

Types (σ, τ, μ) are obtained with the grammar $\sigma ::= \iota \mid o \mid \sigma\sigma$. The elements of o are the two truth values, ι is interpreted as a nonempty set, and a function type $\sigma\tau$ is interpreted as the set of all total functions from σ to τ.

We assume a countable set of *parameters* (x), where every parameter comes with a unique type, and where for every type there are infinitely many parameters of this type. We employ the *logical constants* $\bot : o$, $\neg : oo$, $\wedge : ooo$ and $=_\sigma : \sigma\sigma o$, where there is a logical constant $=_\sigma$ for every type σ. The logical constants take their standard interpretation. *Terms* (s, t, u, v) are defined inductively such that every term has a unique type: (1) every parameter is a term; (2) every logical constant is a term; (3) if s is a term of type $\tau\mu$ and t is a term of type τ, then st is a term of type μ; (4) if x is a name of type σ and t is a term of type τ, then $\lambda x.t$ is a term of type $\sigma\tau$. We write $s : \sigma$ to say that s is a term of type σ. If T is a set of terms, T^σ denotes the set of all terms that are in T and have type σ.

The logical constants $=_\sigma$ are called *identities*, and terms of type o are called *formulas*. Formulas of the form $s =_\sigma t$ are called *equations*, and formulas of the form $\neg(s =_\sigma t)$ are called *disequations*. We write disequations as $s \neq_\sigma t$. We usually write equations and disequations without the type index σ.

A term is *basic* if it contains no other identity but $=_\iota$. We write Λ_σ for the set of all basic terms of type σ. A formula is *normal* if it is a basic formula or a disequation $s \neq t$ where s and t are basic terms. A *normal set* is a set of normal formulas.

The definition of normal formulas is asymmetric in that equations are restricted to type ι while disequations $s \neq_\sigma t$ are allowed at any type σ. The reason for this asymmetry is that the tableau system uses disequations as internal workhorse. Since $s{\neq}t$ and $ps \wedge \neg pt$ are equisatisfiable if p is fresh, normal formulas are not more expressive than basic formulas.

The definition of basic formulas can be extended with further propositional connectives including $=_o$. Since they can be expressed with the connectives we already have, this does not buy new expressivity.

$$\textsc{bot}_\neg \quad \frac{s\,,\,\neg s}{\bot} \qquad \textsc{dn} \quad \frac{\neg\neg s}{s} \qquad \textsc{and} \quad \frac{s \wedge t}{s\,,\,t} \qquad \textsc{and}_\neg \quad \frac{\neg(s \wedge t)}{\neg s \mid \neg t}$$

$$\textsc{mat} \quad \frac{xs_1 \dots s_n\,,\,\neg xt_1 \dots t_n}{s_1 \neq t_1 \mid \cdots \mid s_n \neq t_n} \qquad \textsc{dec} \quad \frac{xs_1 \dots s_n \neq_\iota xt_1 \dots t_n}{s_1 \neq t_1 \mid \cdots \mid s_n \neq t_n}$$

$$\textsc{bot}_{\neq} \quad \frac{s \neq s}{\bot} \qquad \textsc{be} \quad \frac{s \neq_o t}{s\,,\,\neg t \;\mid\; \neg s\,,\,t}$$

$$\textsc{fe} \quad \frac{s \neq_{\sigma\tau} t}{sx \neq tx} \quad x : \sigma \text{ fresh and } s{\neq}t \text{ not evident in } A$$

$$\textsc{sym} \quad \frac{s =_\iota t}{t = s} \qquad \textsc{con} \quad \frac{s =_\iota t\,,\,u \neq_\iota v}{s \neq u \mid t \neq v}$$

A is the normal set to which the rule is applied
x fresh means that x does not occur in A
MAT and DEC assume $n \geq 1$

Fig. 1. Tableau system $\mathcal{B}$

For simplicity we provide only one base type ι different from o. Everything generalizes to countably many base types.

3 Tableau System

Figure 1 shows the rules of a terminating tableau system $\mathcal{B}$ that decides the satisfiability of finite normal sets. For the application constraint of Rule FE we supply the following definition. A disequation $s \neq_{\sigma\tau} t$ is *evident in* A if there exist $n \geq 1$ basic terms $u_1, \dots, u_n$ such that $su_1 \dots u_n \neq tu_1 \dots u_n$ or $tu_1 \dots u_n \neq su_1 \dots u_n$ is in A. The names of the rules are derived as follows: MAT for Mating, FE for functional extensionality, BE for Boolean extensionality, DEC for decomposition, and CON for confrontation.

The rules in the first line of Figure 1 are the usual tableau rules deciding propositional logic. They also decide quantifier-free first-order logic without equality. In contrast to classical first-order logic, type theory allows *embedded formulas*, for instance $p(\neg x)$ where $p : oo$ and $x : o$. The rules MAT, DEC and BE handle embedded formulas. MAT decomposes "atomic" formulas into disequations, which are further decomposed with DEC. Embedded formulas are then fed back to the propositional rules by BE, as demonstrated by the following example.

Example 3.1. The following tableau refutes an unsatisfiable normal set with embedded formulas.

$$\begin{array}{c}
p(fx(\neg\neg y)),\ \neg p(fxy)\\
\text{MAT}\\
fx(\neg\neg y)\neq fxy\\
\text{DEC}\\
\hline
\begin{array}{c|c}
\begin{array}{c} x\neq x\\ \text{BOT}_{\neq}\\ \bot \end{array} &
\begin{array}{c} \neg\neg y\neq y\\ \text{BE}\\ \hline
\begin{array}{c|c}
\neg\neg y,\ \neg y & \neg\neg\neg y,\ y\\
\text{BOT}_{\neg} & \text{DN}\\
\bot & \neg y\\
 & \text{BOT}_{\neg}\\
 & \bot
\end{array}
\end{array}
\end{array}
\end{array}$$

The types of the parameters are $p : \iota o$, $f : \iota o \iota$, $x : \iota$, and $y : o$. □

Example 3.2. Rules SYM and CON handle positive equations at ι. This is demonstrated by the following refutation.

$$\begin{array}{c}
a=b,\ fa=gb,\ fb\neq ga\\
\text{CON}\\
\hline
\begin{array}{c|c}
 & gb\neq ga\\
fa\neq fb & \text{DEC}\\
\text{DEC} & b\neq a\\
a\neq b & \text{SYM}\\
\text{BOT}_{\neg} & b=a\\
\bot & \text{BOT}_{\neg}\\
 & \bot
\end{array}
\end{array}$$

The types of the parameters are $a, b : \iota$ and $f, g : \iota\iota$. □

The confrontation rule CON does not exist in first-order systems. First-order systems typically employ the replacement rule

$$\text{REP}\quad \frac{s =_\iota t\ ,\ C[s]}{C[t]}$$

Example 3.2 can also be refuted with the replacement rule instead of the confrontation rule. However, the confrontation rule is more powerful than the replacement rule since it supports the decomposition needed for embedded formulas. This is illustrated by the next example, which cannot be refuted with the replacement rule.

Example 3.3. Consider the normal set

$$\begin{array}{c} fa = gb,\ f'a = g'c,\ f''b = g''c\\ fb \neq ga,\ f'c \neq g'a,\ f''c \neq g''b \end{array}$$

with the typing $a, b : o$ and $f, g, f', g', f'', g'' : o\iota$. The replacement rule REP cannot be applied to the set. The confrontation rule CON can be applied to 3 confrontation pairs. Application of DEC now yields 8 sets that all contain the unsatisfiable set

$$a \neq b,\ a \neq c,\ b \neq c$$

up to symmetry. This set can be refuted with BE and BOT$_{\neg}$. Thus the initial set can be refuted with $\mathcal{B}$. □

It remains to illustrate the use of the functional extensionality rule FE. FE is only needed if higher-order parameters are present.

Example 3.4. The following tableau has two branches both of which contain the refutable set $\{a \neq b,\ a \neq c,\ b \neq c\}$. Thus the set in the first line is refutable.

$$\begin{array}{c}
a \neq b,\ fa,\ fb,\ ga,\ gb,\ pf,\ \neg pg \\
\textsc{mat} \\
f \neq g \\
\textsc{fe} \\
fc \neq gc \\
\textsc{be} \\
\hline
\begin{array}{c|c}
fc,\ \neg gc & \neg fc,\ gc \\
\textsc{mat} & \textsc{mat} \\
a \neq c & a \neq c \\
\textsc{mat} & \textsc{mat} \\
b \neq c & b \neq c
\end{array}
\end{array}$$

Note the crucial use of the functional extensionality rule FE. The types of the parameters are $a, b, c : o,\ \ f, g : oo$ and $p : (oo)o$. □

In summary we can now say that $\mathcal{B}$ extends the classical propositional system with the rules MAT, DEC, BOT$_{\neq}$, BE, FE and CON to account for embedded formulas. MAT and DEC decompose formulas that are atomic for the classical rules. This way BE can finally lift embedded formulas to the top level. To deal with equality, the traditional replacement rule is replaced by the confrontation rule. All rules so far are already needed for first-order normal formulas. For higher-order parameters a single rule FE incorporating functional extensionality is needed.

The only higher-order tableau system we could find in the literature is the calculus of Kohlhase [12]. Kohlhase's calculus does not have equality, but there are unification constraints that play the role of our top level disequations. For unification constraints Kohlhase has rules that are similiar to MAT, DEC, FE, and BE. Our tableau rules also have similarities with the rules in Benzmüller's [13] higher-order RUE-resolution calculus, which employs primitive equality. In particular, the RUE calculus allows resolution of positive equations against negative equations (which play the role of unification constraints). Combining this form of resolution with decomposition, one obtains a rule

$$\frac{C \vee s = t \qquad D \vee u \neq v}{C \vee D \vee s \neq u \vee t \neq v}$$

which is essentially the same as our confrontation rule CON.

4 Soundness and Termination

The rules in Figure 1 apply to normal sets and produce one or several normal sets by adding normal formulas. More precisely, if a rule applies to a normal set A, it yields $n \geq 1$ normal sets $A_1, \dots, A_n$ called *alternatives* such that $A \subseteq A_i$ for all $i \in \{1, \dots, n\}$. If $n \geq 2$, the rule applied is called *branching*. To obtain a terminating system, we admit only applications where $\perp \notin A$ and the alternatives $A_1, \dots, A_n$ are all proper supersets of A (i.e., $A \subsetneq A_i$). The tableau system $\mathcal{B}$ is *sound* if for every application of a rule the set A is satisfiable if and only if one of the alternatives $A_1, \dots, A_n$ is satisfiable. For the termination of $\mathcal{B}$ we consider the relation $A \to A'$ on normal sets that holds if and only if A' can be obtained as an alternative by a rule that applies to A. We say that $\mathcal{B}$ *terminates* if this relation terminates on finite normal sets. Finally, we call a normal set A *evident* if $\perp \notin A$ and no rule of $\mathcal{B}$ applies to A. We will show the following:

- $\mathcal{B}$ is sound.
- $\mathcal{B}$ terminates on finite normal sets.
- Evident sets are satisfiable, and finite evident sets are finitely satisfiable.

Together, soundness, termination and model existence yield a decision procedure for the satisfiability of finite normal sets.

Proposition 4.1 (Soundness). *$\mathcal{B}$ is sound.*

Proof. Let $A_1, \dots, A_n$ be obtained from A by application of a rule. It suffices to show that for every model of A there exists an interpretation that is a model of at least one of the alternatives $A_1, \dots, A_n$. For BOT$_\neg$ this follows from the fact that the implication $x \wedge \neg x \to \perp$ is valid. For AND$_\neg$ the validity of $\neg(x \wedge y) \to \neg x \vee \neg y$ suffices, and for FE the validity of $f{\neq}g \to \exists x. fx{\neq}gx$ does the job. Note that this is in fact equivalent to functional extensionality $(\forall x. fx{=}gx) \to f{=}g$. The soundness of the other rules follows with similar arguments. □

For the termination proof we distinguish between Rule FE and the other rules. FE is special in that it introduces new parameters. We first show that the subsystem $\mathcal{B}_0$ of $\mathcal{B}$ obtained by removing FE terminates. The proof will exhibit an upper bound function $\mathcal{U}$ from sets of terms to sets of terms such that the following holds for every derivation $A_1 \to \dots \to A_n$ in $\mathcal{B}_0$:

1. $\mathcal{U}A_1 = \dots = \mathcal{U}A_n$
2. $\mathcal{U}A_i$ is a finite set such that $A_i \subseteq \mathcal{U}A_i$ for all $i = 1, \dots, n$.

Since $A_1 \subsetneq \dots \subsetneq A_n$, the bound function suffices for termination of $\mathcal{B}_0$.

Let T range over sets of terms. We define the bound function as $\mathcal{U}T := \mathcal{C}(\mathcal{S}(\mathcal{E}T))$ where the functions $\mathcal{S}$ (subterms), $\mathcal{E}$ (elements) and $\mathcal{C}$ (compounds) are defined as follows:

- $\mathcal{E}T$ is the least set of terms such that:
  1. $\perp \in \mathcal{E}T$
  2. If $(s =_\iota t) \in T$, then $s, t \in \mathcal{E}T$.

   3. If $(s \neq t) \in T$, then $s, t \in \mathcal{E}T$.
   4. If $\neg s \in T$ and s is not an equation, then $s \in \mathcal{E}T$.
   5. If $s \in T$ and s is neither a negated term nor an equation, then $s \in \mathcal{E}T$.
- $\mathcal{S}T$ is the set of all subterms of the terms in T.
- $\mathcal{C}T$ is the least set of terms such that:
   1. $T \subseteq \mathcal{C}T$.
   2. If $s, t \in T^\iota$, then $(s =_\iota t) \in \mathcal{C}T$.
   3. If $s, t \in T^\sigma$, then $(s \neq t) \in \mathcal{C}T$.
   4. If $s \in T^o$, then $\neg s \in \mathcal{C}T$.

All three functions are monotone functions from set of terms to set of terms that preserve finiteness. The following properties are easy to verify:

1. If $A \rightarrow A'$ in $\mathcal{B}_0$, then $\mathcal{S}(\mathcal{E}A) = \mathcal{S}(\mathcal{E}A')$.
2. $T \subseteq \mathcal{C}(\mathcal{E}T) \subseteq \mathcal{C}(\mathcal{S}(\mathcal{E}T))$.

Hence $\mathcal{U}$ has the required properties and $\mathcal{B}_0$ terminates.

We now extend the termination result to $\mathcal{B}$. We define the *power of* A as the multiset that contains for each function type σ as many copies of σ as there are 2-element subsets $\{s, t\} \subseteq (\mathcal{S}(\mathcal{E}A))^\sigma$ such that $s \neq t$ is not evident in A. It is not difficult to verify the following for normal sets A:

1. Application of Rule FE decreases the power of A.
2. Application of a rule other than FE does not increase the power of A.

Hence we have proved the termination of $\mathcal{B}$.

Proposition 4.2 (Termination). *$\mathcal{B}$ terminates.*

5 Model Existence

Recall that an evident set is a normal set that does not contain $\bot$ and to which no rule of $\mathcal{B}$ can add a formula.

Theorem 5.1 (Model Existence). *Every evident set has a model, and every finite evident set has a finite model.*

Before proving the theorem we state two important consequences.

Corollary 5.2. *The tableau system $\mathcal{B}$ constitutes a decision procedure for the satisfiability of normal formulas.*

Proof. Follows from Theorem 5.1 since $\mathcal{B}$ is sound and terminating. □

Corollary 5.3. *Normal formulas have the finite model property.*

Proof. Let s be a satisfiable normal formula. Since $\mathcal{B}$ is sound and terminating, we can obtain a finite evident set E that contains s. Now Theorem 5.1 gives us a finite model of E and hence of s. □

We now begin the proof of the model existence theorem. Let E be an evident set. We will construct a model $\mathcal{I}$ of E that is finite if E is finite. We arrange the following:

- $\mathcal{I}o := \{0,1\}$
- $\mathcal{I}(\sigma\tau) :=$ set of all total functions from $\mathcal{I}\sigma$ to $\mathcal{I}\tau$
- $\mathcal{I}\bot := 0$
- $\mathcal{I}(\neg)$, $\mathcal{I}(\wedge)$, and $\mathcal{I}(=_\sigma)$ are defined as usual.

It remains to define $\mathcal{I}$ for the type ι and the parameters.

Discriminants

We prepare the definition of $\mathcal{I}\iota$. We write $s\sharp t$ if E contains $s{\neq}t$ or $t{\neq}s$. A term $s \in \Lambda_\iota$ is *discriminating* if there is a term t such that $s\sharp t$. We write Δ for the set of all discriminating terms. A *discriminant* is a maximal subset $D \subseteq \Delta$ such that for all $s\sharp t$ either $s \notin D$ or $t \notin D$. We define $\mathcal{I}\iota$ to be the set of all discriminants.

- $\mathcal{I}\iota := \{\, D \mid D \text{ is a discriminant}\,\}$

Example 5.4. Suppose $E = \{x{\neq}y,\ x{\neq}z,\ y{\neq}z\}$ and $x, y, z : \iota$. Then there are 3 discriminants: $\{x\}$, $\{y\}$, $\{z\}$. □

Example 5.5. Suppose $E = \{x{\neq}f(fx),\ fx{\neq}f(f(fx))\}$ and $f : \iota\iota$. Then there are 4 discriminants: $\{x, fx\}$, $\{x, f(f(fx))\}$, $\{f(fx), fx\}$, $\{f(fx), f(f(fx))\}$. □

Proposition 5.6. *If E contains exactly n disequations at ι, then there are at most 2^n discriminants. If E contains no disequation at ι, then $\emptyset$ is the only discriminant.*

Proposition 5.7. *Let D_1 and D_2 be different discriminants. Then:*

1. *D_1 and D_2 are separated by a disequation in E, that is, there exist terms $t_1 \in D_1$ and $t_2 \in D_2$ such that $t_1\sharp t_2$.*
2. *D_1 and D_2 are not connected by an equation in E, that is, there exist no terms $t_1 \in D_1$ and $t_2 \in D_2$ such that $(t_1{=}t_2) \in E$.*

Proof. The first claim follows by contradiction. Suppose there are no terms $t_1 \in D_1$ and $t_2 \in D_2$ such that $t_1\sharp t_2$. Let $t \in D_1$. Then $t \in D_2$ since D_2 is a maximal compatible set of discriminating terms. Thus $D_1 \subseteq D_2$. A symmetric argument yields $D_2 \subseteq D_1$. Contradiction.

The second claim also follows by contradiction. Suppose there is an equation $(s_1{=}s_2) \in E$ such that $s_1 \in D_1$ and $s_2 \in D_2$. By the first claim we have terms $t_1 \in D_1$ and $t_2 \in D_2$ such that $t_1\sharp t_2$. Since E is closed under CON, we have $s_1\sharp t_1$ or $s_2\sharp t_2$. Contradiction since D_1 and D_2 are discriminants. □

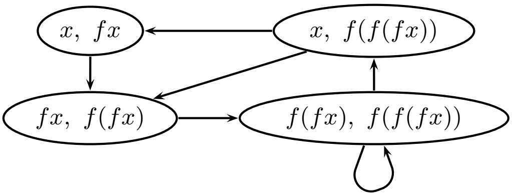
Possible Values

It remains to define $\mathcal{I}$ for the parameters. Given the presence of higher-order parameters this is not straightforward. We base the definition on a family of relations $\triangleright_\sigma \subseteq \Lambda_\sigma \times \mathcal{I}\sigma$ defined by induction on types:

$$\begin{aligned} s \triangleright_o 0 &\;:\Longleftrightarrow\; s \notin E \\ s \triangleright_o 1 &\;:\Longleftrightarrow\; \neg s \notin E \\ s \triangleright_\iota D &\;:\Longleftrightarrow\; s \in [D] \\ s \triangleright_{\sigma\tau} f &\;:\Longleftrightarrow\; st \triangleright_\tau fa \text{ whenever } t \triangleright_\sigma a \\ [D] &\;:=\; D \cup \{\, s \in \Lambda_\iota \mid s \text{ not discriminating} \,\} \end{aligned}$$

We read $s \triangleright a$ as "s can be a" or "a is a *possible value* for s". Note that if s is a basic formula such that $s \notin E$ and $\neg s \notin E$, then both 0 and 1 are possible values for s. We will show that every basic term has a possible value and that we obtain a model of E if we define $\mathcal{I}x$ as a possible value for x for every parameter x. Such a model will satisfy $s \triangleright \hat{\mathcal{I}}s$ for every basic term s. Note that $\hat{\mathcal{I}}s$ denotes the value the term s evaluates to in the model $\mathcal{I}$.

Example 5.8. Suppose $E = \{x{\neq}f(fx),\ fx{\neq}f(f(fx))\}$ and $f : \iota\iota$. The following graph shows the discriminants and the possible pairs for possible values of f.



There are 2 possible values for x. To obtain a possible value for f, we must choose for every node exactly one departing edge. Hence there are 4 possible values for f. For each choice of a value for x and f we obtain a model of E. Altogether we obtain 8 models this way. Four of the obtained models have a junk value at ι (i.e., a value that is not denoted by a basic term). From the models with a junk value we can obtain three-valued models. There is no two-valued model. □

Compatibility

We define a family of relations $\|_\sigma \subseteq \Lambda_\sigma \times \Lambda_\sigma$ by induction on types:

$$\begin{aligned} s \|_o t &\;:\Longleftrightarrow\; \{s, \neg t\} \not\subseteq E \text{ and } \{\neg s, t\} \not\subseteq E \\ s \|_\iota t &\;:\Longleftrightarrow\; \text{not } s \sharp t \\ s \|_{\sigma\tau} t &\;:\Longleftrightarrow\; su \|_\tau tv \text{ whenever } u \|_\sigma v \end{aligned}$$

We say that s and t are *compatible* if $s \| t$. A set T of terms is *compatible* if $s \| t$ for all terms $s, t \in T$. If $T \subseteq \Lambda_\sigma$, we write $T \triangleright a$ if a is a common possible value for all terms $s \in T$. We will show that a set of equi-typed terms is compatible if and only if all its terms have a common possible value.

Proposition 5.9. *The compatibility relations $\|_\sigma$ are symmetric.*

The compatibility relations are also reflexive. Showing this fact will take some effort. For the induction to go through we will strengthen the hypothesis. First we prove the following lemma.

Lemma 5.10. *Let s be a basic term. Then:*

1. *If $s : o$, then $s \parallel s$.*
2. *If $s = cs_1 \dots s_n$ and c is a logical constant, then $s \parallel s$.*

Proof. The first claim follows by contradiction. Suppose $s \nparallel_o s$. Then $s, \neg s \in E$, contradicting the assumption that E is evident.

Given that the first claim holds, for the second claim it suffices to consider terms of the forms $\neg$, $\wedge$, $=_\iota$, $(\wedge)t$, and $(=_\iota)t$. In all cases the claim follows by contradiction. We show $=_\iota \parallel =_\iota$. The other cases are similar.

Suppose $=_\iota \nparallel =_\iota$. Then there exist terms such that $s_1 \parallel_\iota s_2$, $t_1 \parallel_\iota t_2$, and $s_1{=_\iota}t_1 \nparallel_o s_2{=_\iota}t_2$. Then either $s_1{=}t_1$ and $s_2{\neq}t_2$ are both in E or $s_1{\neq}t_1$ and $s_2{=}t_2$ are both in E. Since E is closed under CON, we have either $s_1 \sharp s_2$ (contradicting $s_1 \parallel s_2$) or $t_1 \sharp t_2$ (contradicting $t_1 \parallel t_2$). □

Lemma 5.11 (Reflexivity). *For every type σ and all basic terms s, t, $xs_1 \dots s_n$, $xt_1 \dots t_n$ of type σ:*

1. *We never have both $s \parallel_\sigma t$ and $s \sharp t$.*
2. *We always have either $xs_1 \dots s_n \parallel_\sigma xt_1 \dots t_n$ or $s_i \sharp t_i$ for some $i \in \{1, \dots, n\}$.*
3. *We always have $s \parallel_\sigma s$.*

Proof. By mutual induction on σ. The base cases for Claim (1) follow easily from the definition of compatibility and closure of E under BE. The base cases for Claim (2) use closure of E under MAT and DEC, and the base cases for Claim (3) use closure of E under BOT$_\neg$ and BOT$_{\neq}$. Next we show the claims for $\sigma = \tau\mu$.

1. By contradiction. Suppose $s \parallel_\sigma t$ and $s \sharp t$. Since E is closed under FE there exist $n \geq 1$ terms $u_1, \dots, u_n$ such that $su_1 \dots u_n \sharp tu_1 \dots u_n$. By inductive hypothesis (3) we have $u_i \parallel u_i$ for $i = 1, \dots, n$. Hence $su_1 \dots u_n \parallel tu_1 \dots u_n$ since $s \parallel_\sigma t$. This contradicts inductive hypothesis (1) since $su_1 \dots u_n \sharp tu_1 \dots u_n$.

2. Suppose $xs_1 \dots s_n \nparallel_\sigma xt_1 \dots t_n$. Then there exist terms $u \parallel_\tau v$ such that $xs_1 \dots s_n u \nparallel_\mu xt_1 \dots t_n v$. By inductive hypotheses (2) and (1) we have $s_i \sharp t_i$ for some $i \in \{1, \dots, n\}$.

3. Suppose $s \nparallel_\sigma s$. By Lemma 5.10 we have $s = xs_1 \dots s_n$. By Claim (2), which we have already established for σ, we have $s_i \sharp s_i$ for some $i \in \{1, \dots, n\}$. Contradiction since E is closed under BOT$_{\neq}$. □

Lemma 5.12 (Common Value). *Let $T \subseteq \Lambda_\sigma$. Then T is compatible if and only if there exists a value a such that $T \rhd_\sigma a$.*

Proof. By induction on σ.

$\sigma = o$, $\Rightarrow$. By contraposition. Suppose $T \ntriangleright 0$ and $T \ntriangleright 1$. Then there are terms $s, t \in T$ such that $s, \neg t \in E$. Thus $s \nparallel t$. Hence T is not compatible.

$\sigma = o$, $\Leftarrow$. By contraposition. Suppose $s \nparallel_o t$ for $s, t \in T$. Then $s, \neg t \in E$ without loss of generality. Hence $s \ntriangleright 0$ and $t \ntriangleright 1$. Thus $T \ntriangleright 0$ and $T \ntriangleright 1$.

$\sigma = \iota$, $\Rightarrow$. Let T be compatible. Then there exists a discriminant D that contains all the discriminating terms in T. Thus $T \triangleright D$.

$\sigma = \iota$, $\Leftarrow$. By contradiction. Suppose $T \triangleright D$ and T is not compatible. Then there are terms $s, t \in T$ such that $s \sharp t$. Thus s and t cannot be both in D. This contradicts $s, t \in T \triangleright D$.

$\sigma = \tau\mu$, $\Rightarrow$. Let T be compatible. We define $T_a := \{\, ts \mid t \in T,\ s \triangleright_\tau a \,\}$ for every value $a \in \mathcal{I}\tau$ and show that T_a is compatible. Let $t_1, t_2 \in T$ and $s_1, s_2 \triangleright_\tau a$. It suffices to show $t_1 s_1 \parallel t_2 s_2$. By the inductive hypothesis $s_1 \parallel_\tau s_2$. Since T is compatible, $t_1 \parallel t_2$. Hence $t_1 s_1 \parallel t_2 s_2$.

By the inductive hypothesis we now know that for every $a \in \mathcal{I}\tau$ there is a $b \in \mathcal{I}\mu$ such that $T_a \triangleright_\mu b$. Hence there is a function $f \in \mathcal{I}\sigma$ such that $T_a \triangleright_\mu fa$. Thus $T \triangleright_\sigma f$.

$\sigma = \tau\mu$, $\Leftarrow$. Let $T \triangleright_\sigma f$ and $s, t \in T$. We show $s \parallel_\sigma t$. Let $u \parallel_\tau v$. It suffices to show $su \parallel_\mu tv$. By the inductive hypothesis $u, v \triangleright_\tau a$ for some value a. Hence $su, tv \triangleright_\mu fa$. Thus $su \parallel_\mu tv$ by the inductive hypothesis. □

By Lemmas 5.11 and 5.12 we have a possible value for every parameter x (since $x \parallel x$). Hence we can define $\mathcal{I}x$ such that $x \triangleright \mathcal{I}x$ for all parameters x. This completes the definition of the interpretation $\mathcal{I}$.

Lemma 5.13 (Admissibility). *For all basic terms s: $s \triangleright \hat{\mathcal{I}}s$.*

Proof. By induction on s. Let s be a basic term.

If $s = x$ is a parameter, we haven chosen $\mathcal{I}x$ such that $x \triangleright \mathcal{I}x$.

If $s = tu$ is an application, we have $t \triangleright \hat{\mathcal{I}}t$ and $u \triangleright \hat{\mathcal{I}}u$ by the inductive hypothesis. Hence $tu \triangleright (\hat{\mathcal{I}}t)(\hat{\mathcal{I}}u) = \hat{\mathcal{I}}(tu)$.

Let $s = (\wedge)$. Assume $t_1 \triangleright_o a$ and $t_2 \triangleright_o b$. We show $t_1 \wedge t_2 \triangleright \mathcal{I}(\wedge)ab$ by contradiction. Suppose $t_1 \wedge t_2 \ntriangleright \mathcal{I}(\wedge)ab$. Case analysis.

1. $a = b = 1$. Then $\neg t_1, \neg t_2 \notin E$ and $\neg(t_1 \wedge t_2) \in E$. Contradiction since E is closed under AND$_\neg$.
2. $a = 0$ or $b = 0$. Then $t_1 \notin E$ or $t_2 \notin E$, and $(t_1 \wedge t_2) \in E$. Contradiction since E is closed under AND.

Let $s = (=_\iota)$. Assume $t_1 \triangleright_\iota D_1$ and $t_2 \triangleright_\iota D_2$. We show $(t_1 = t_2) \triangleright_o \mathcal{I}(=_\iota)D_1 D_2$ by contradiction. Suppose $(t_1 = t_2) \ntriangleright_o \mathcal{I}(=_\iota)D_1 D_2$. Case analysis.

1. $D_1 = D_2$. Then $(t_1 \neq t_2) \in E$. Hence t_1, t_2 are discriminating and thus $t_1 \in D_1$ and $t_2 \in D_2 = D_1$. Contradiction by the definition of discriminants.
2. $D_1 \neq D_2$. Then $(t_1 = t_2) \in E$. Contradiction by Proposition 5.7(2).

The case $s = \neg$ follows with the closure of E under DN. The case $s = \bot$ is straightforward. □

Lemma 5.14. *For all formulas $s \in E$: $\hat{\mathcal{I}}s = 1$.*

Proof. Let $s \in E$. Then s is either basic or a disequation between basic terms.

Suppose s is basic. By Lemma 5.13, $s \triangleright \hat{\mathcal{I}}s$. On the other hand, $s \not\triangleright 0$ since $s \in E$. Hence $\hat{\mathcal{I}}s = 1$.

Suppose $s = (s_1 {\neq} s_2)$ where s_1 and s_2 are basic. Then $s_1 \sharp s_2$. Assume $\hat{\mathcal{I}}s_1 = \hat{\mathcal{I}}s_2$. Then $s_1, s_2 \triangleright \hat{\mathcal{I}}s_1$ by Lemma 5.13 and hence $s_1 \parallel s_2$ by Lemma 5.12. Contradiction by Lemma 5.11(1) since $s_1 \sharp s_2$. □

This completes the proof of Theorem 5.1.

6 Extensions

A Henkin-complete cut-free tableau system $\mathcal{T}$ for extensional type theory can be obtained as the dual of Brown's one-sided sequent system [9]. Our system $\mathcal{B}$ is in fact a subsystem of this system. $\mathcal{B}$ contains all the distinctive rules of $\mathcal{T}$ (MAT, DEC, CON). In the following, we consider the additional rules of $\mathcal{T}$ and discuss their impact on termination.

General Equality

$\mathcal{B}$ restricts equality to the base type ι. If we admit all identities in basic terms, two additional rules are needed:

$$\text{EQB}\ \frac{s =_o t}{s\,,\, t \;\mid\; \neg s\,,\, \neg t} \qquad\qquad \text{EQF}\ \frac{s =_{\sigma\tau} t}{su = tu}$$

Rule EQF destroys termination. However, we can preserve termination if we restrict EQF such that u must be a term that already occurs as a subterm. It is open whether the resulting system is complete.

Lambda Abstraction

$\mathcal{B}$ disallows lambda abstractions. If we admit lambda abstractions in basic terms, an additional rule incorporating β-reduction is needed:

$$\text{BETA}\ \frac{s}{t} \quad t \text{ is obtainable from } s \text{ by } \beta\text{-reduction}$$

Example 6.1. $\mathcal{B}$ extended with BETA can prove the η-law:

$(\lambda x.fx) \neq f$	initial formula
$(\lambda x.fx)a \neq fa$	FE
$fa \neq fa$	BETA
$\bot$	BOT$_{\neq}$

□

Example 6.2. $\mathcal{B}$ extended with BETA does not terminate:

$p(\lambda x.p(fx)),\ \neg p(fa)$	initial formulas	$x, a : \sigma,\quad f : \sigma\sigma o,\quad p : (\sigma o)o$
$(\lambda x.p(fx)) \neq fa$	MAT	
$(\lambda x.p(fx))b \neq fab$	FE	
$p(fb) \neq fab$	BETA	
$\neg p(fb),\ fab$	BE	
$(\lambda x.p(fx)) \neq fb$	MAT	
...		

Note that σ can be any type. The problem are the new parameters introduced by FE and the disequations introduced by MAT. There seems to be no easy fix. For $\sigma = \iota$ the initial formulas do have a finite model: $\mathcal{I}\iota = \mathcal{I}o$, $\mathcal{I}f = \lambda xy.y$, $\mathcal{I}p(\lambda x.x) = 0$, $\mathcal{I}p(\lambda x.0) = 1$. □

Quantifiers

The extension of $\mathcal{B}$ with general equality and lambda abstraction can express quantification and thus already covers full simple type theory. If desired, quantification can be directly accounted for by additional logical constants. For universal quantification, we may have the constants $\forall_\sigma : (\sigma o)o$ and the rules

$$\text{ALL}\quad \frac{\forall_\sigma s}{st}\ \ t : \sigma \qquad\qquad \text{ALL}_{\neg}\quad \frac{\neg\forall_\sigma s}{\neg sx}\ \ x : \sigma \text{ fresh}$$

7 Conclusion

We have presented a terminating tableau system that decides satisfiability of basic formulas with respect to standard models. This contributes a new decidability and completeness result for higher-order logic with standard semantics. Our model existence proof relies on the possible-values technique, which for the first time is used to construct standard models. We are confident that our results can be extended. On the one side, the addition of equations at functional types may preserve decidability. On the other side, the addition of first-order quantifiers may preserve completeness.

Besides theoretical curiosity, there is a practical interest behind our research. We feel that the decision technique presented in this paper will lead to stronger auto tactics for interactive theorem provers. Even with a naive implementation, our decision technique can decide many problems that are out of the reach of current systems. As is, decomposition and confrontation may cause combinatorial explosion. An idea for further research is the integration of congruence closure techniques [14], which could efficiently replace most applications of the branching confrontation rule.

References

1. Andrews, P.B.: Classical type theory. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. 2, pp. 965–1007. Elsevier Science, Amsterdam (2001)
2. Farmer, W.M.: The seven virtues of simple type theory. J. Appl. Log. 6(3), 267–286 (2008)
3. Friedman, H.: Equality between functionals. In: Parikh, R. (ed.) Proc. Logic Colloquium 1972-73. Lectures Notes in Mathematics, vol. 453, pp. 22–37. Springer, Heidelberg (1975)
4. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences 18, 194–211 (1979)
5. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science 27, 333–354 (1983)
6. Nipkow, T., Paulson, L.C., Wenzel, M.T.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
7. Andrews, P.B., Brown, C.E.: TPS: A hybrid automatic-interactive system for developing proofs. Journal of Applied Logic 4(4), 367–395 (2006)
8. Benzmüller, C., Paulson, L.C., Theiss, F., Fietzke, A.: LEO-II - A cooperative automatic theorem prover for classical higher-order logic (System description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS(LNAI), vol. 5195, pp. 162–170. Springer, Heidelberg (2008)
9. Brown, C.E.: Automated Reasoning in Higher-Order Logic: Set Comprehension and Extensionality in Church's Type Theory. College Publications (2007)
10. Takahashi, M.: A proof of cut-elimination theorem in simple type theory. Journal of the Mathematical Society of Japan 19, 399–410 (1967)
11. Prawitz, D.: Hauptsatz for higher order logic. J. Symb. Log. 33, 452–457 (1968)
12. Kohlhase, M.: Higher-order tableaux. In: Baumgartner, P., Posegga, J., Hähnle, R. (eds.) TABLEAUX 1995. LNCS, vol. 918, pp. 294–309. Springer, Heidelberg (1995)
13. Benzmüller, C.: Extensional higher-order paramodulation and RUE-resolution. In: Ganzinger, H. (ed.) CADE 1999. LNCS(LNAI), vol. 1632, pp. 399–413. Springer, Heidelberg (1999)
14. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM 27(2), 356–364 (1980)

Modular Sequent Systems for Modal Logic

Kai Brünnler[1] and Lutz Straßburger[2]

[1] Institut für angewandte Mathematik und Informatik,
Neubrückstr. 10, CH – 3012 Bern, Switzerland
http://www.iam.unibe.ch/ kai/
[2] École Polytechnique, Laboratoire d'Informatique (LIX),
Projet Parsifal, Rue de Saclay, 91128 Palaiseau Cedex, France
http://www.lix.polytechnique.fr/ lutz/

Abstract. We see cut-free sequent systems for the basic normal modal logics formed by any combination the axioms $\mathsf{d}, \mathsf{t}, \mathsf{b}, \mathsf{4}, \mathsf{5}$. These systems are *modular* in the sense that each axiom has a corresponding rule and each combination of these rules is complete for the corresponding frame conditions. The systems are based on *nested sequents*, a natural generalisation of hypersequents. Nested sequents stay inside the modal language, as opposed to both the display calculus and labelled sequents. The completeness proof is via syntactic cut elimination.

1 Introduction

This paper is part of a research effort to develop proof-theoretic systems for modal logic that stay inside the modal language. This requirement in particular excludes the display calculus [2,16], which is formulated in the language of tense logic, and labelled sequents, see for example [12], which are formulated in a hybrid language.

Examples of modal proof systems that stay inside the modal language are hypersequent systems [1], systems in the calculus of structures [10,14,15,9] and, what we will study here, systems using *nested sequents*. Nested sequents are a natural generalisation of hypersequents, and they have been invented several times independently, for example by Bull [6], by Kashima [11], by Brünnler [4,3] (using the name *deep sequent* at the time) and by Poggiolesi (using the name *tree-hypersequent*) [13].

In this paper we provide cut-free sequent systems for the basic normal modal logics formed from the axioms $\mathsf{d}, \mathsf{t}, \mathsf{b}, \mathsf{4}, \mathsf{5}$ which are *modular* in the sense that each modal axiom has a corresponding sequent calculus rule and that each combination of these rules is sound and complete for the corresponding modal logic. To our knowledge, these are the first systems inside the modal language which are modular.

These systems are closely related to the systems introduced in [3], in particular the subsystem for the modal logic K is essentially the same. Modal axioms in [3] were turned into $\Diamond$-rules, that is, rules where the active formula in the conclusion has the connective $\Diamond$ as its main connective. However, these systems are not

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 152–166, 2009.

modular. For example, the logic $\mathsf{S5}$ can be axiomatised in a Hilbert system both by using the axioms $\mathsf{t}, \mathsf{b}, \mathsf{4}$ and by using $\mathsf{t}, \mathsf{5}$. While there is a complete cut-free system for $\mathsf{S5}$ in [3], namely one with the rules $\mathsf{t}, \mathsf{b}, \mathsf{4}, \mathsf{5}$, neither the cut-free system with rules $\mathsf{t}, \mathsf{b}, \mathsf{4}$ nor the cut-free system with the rules $\mathsf{t}, \mathsf{5}$ are complete for $\mathsf{S5}$. Here, we follow another approach to designing modal rules. We do not turn modal axioms into $\Diamond$-rules, but into structural rules. These structural rules are the key to modularity, as was already conjectured in [4].

Proving completeness for our systems turned out to be challenging. We do so by way of embedding a corresponding Hilbert system and syntactically proving cut-elimination. The cut-elimination proof is interesting: it relies on a decomposition of the contraction rule, similar to what has been observed in deep inference systems for propositional logic, where contraction is decomposed into an atomic version and a local *medial* rule [5].

2 The Sequent Systems

Formulas and modal axioms. Propositions p and their negations $\bar{p}$ are *atoms*, with $\bar{\bar{p}}$ defined to be p. *Formulas*, denoted by A, B, C, D are given by the grammar

$$A ::= p \mid \bar{p} \mid (A \vee A) \mid (A \wedge A) \mid \Diamond A \mid \Box A \quad .$$

Given a formula A, its *negation* $\bar{A}$ is defined as usual using the De Morgan laws, $A \supset B$ is defined as $\bar{A} \vee B$ and $\top$ and $\bot$ are respectively defined as $p \vee \bar{p}$ and $p \wedge \bar{p}$ for some proposition p. Each name in $\{\mathsf{k}, \mathsf{d}, \mathsf{t}, \mathsf{b}, \mathsf{4}, \mathsf{5}\}$ corresponds both to a frame condition and to a Hilbert-style axiom:

k:	$\top$	$\Box(A \vee B) \supset (\Box A \vee \Diamond B)$	b:	symmetric	$A \supset \Box\Diamond A$
d:	serial	$\Box A \supset \Diamond A$	$\mathsf{4}$:	transitive	$\Box A \supset \Box\Box A$
t:	reflexive	$A \supset \Diamond A$	$\mathsf{5}$:	euclidean	$\Diamond A \supset \Box\Diamond A$.

Nested sequents. A *nested sequent* is a finite multiset of formulas and boxed sequents. A *boxed sequent* is an expression $[\Gamma]$ where Γ is a nested sequent. In the following, a *sequent* is a nested sequent. Sequents are denoted by $\Gamma, \Delta, \Lambda, \Pi, \Sigma$. As usual, sequents are written without curly braces and the comma in the expression Γ, Δ is multiset union. A sequent is always of the form

$$A_1, \ldots, A_m, [\Delta_1], \ldots, [\Delta_n] \quad .$$

The *corresponding formula* of the above sequent is $\bot$ if $m = n = 0$ and otherwise

$$A_1 \vee \cdots \vee A_m \vee \Box(D_1) \vee \cdots \vee \Box(D_n) \quad ,$$

where $D_1 \ldots D_n$ are the corresponding formulas of the sequents $\Delta_1 \ldots \Delta_n$. Notice that a sequent induces a tree where each node is marked with a multiset of formulas. We will refer to notions such as the *depth* or the *root* of a sequent, meaning the depth or the root of the corresponding tree.

Sequent contexts. Informally, a context is a sequent with holes. We will mostly encounter sequents with just one or two holes. A *unary context* is a sequent with exactly one occurrence of the symbol $\{\ \}$, the *hole*, which does not occur inside formulas. Such contexts are denoted by $\Gamma\{\ \}$, $\Delta\{\ \}$, and so on. The hole is also called the *empty context*. The sequent $\Gamma\{\Delta\}$ is obtained by replacing $\{\ \}$ inside $\Gamma\{\ \}$ by Δ. For example, if $\Gamma\{\ \} = A, [[B], \{\ \}]$ and $\Delta = C, [D]$ then

$$\Gamma\{\Delta\} = A, [[B], C, [D]] \quad .$$

The *depth* of a unary context $\Gamma\{\ \}$, denoted $\mathit{depth}(\Gamma\{\ \})$ is defined as follows

$$\begin{aligned}&\mathit{depth}(\Gamma, \{\ \}) = 0\\ &\mathit{depth}(\Gamma, [\Delta\{\ \}]) = \mathit{depth}(\Delta\{\ \}) + 1.\end{aligned}$$

More generally, a *context* is a sequent with $n \geq 0$ occurrences of $\{\ \}$, which do not occur inside formulas, and which are linearly ordered. A context with n holes is denoted by

$$\Gamma\underbrace{\{\ \}\ldots\{\ \}}_{n-\text{times}} \quad .$$

Holes can be filled with sequents, or contexts, in general. For example, if $\Gamma\{\ \}\{\ \} = A, [[B], \{\ \}], \{\ \}$ and $\Delta\{\ \} = C, [\{\ \}]$ then

$$\Gamma\{\Delta\{\ \}\}\{\ \} = A, [[B], C, [\{\ \}]], \{\ \} \quad ,$$

where in all contexts the holes are ordered from left to right as shown.

System $\mathsf{K} + \dot{\mathsf{X}}$. Figure 1 shows the set of rules from which we form our deductive systems. *System* K is the set of rules $\{\wedge, \vee, \Box, \mathsf{k}, \mathsf{ctr}\}$. We will look at extensions of System K with sets of rules $\dot{\mathsf{X}} \subseteq \{\dot{\mathsf{d}}, \dot{\mathsf{t}}, \dot{\mathsf{b}}, \dot{\mathsf{4}}, \dot{\mathsf{5}}\}$. The rules in $\dot{\mathsf{X}}$ are called *structural modal rules*. The $\dot{\mathsf{5}}$-rule is a bit special since it uses a two-hole-context. It can actually be decomposed into three rules that use unary contexts. However, here we prefer the presentation with a single rule. The $\dot{\mathsf{5}}$-rule may be understood as allowing to do the following, when going from premise to conclusion: take a boxed sequent $[\Delta]$, which is not at the root of the sequent, and move it to any other place in the sequent.

Notice that we have an explicit contraction rule in system K and that the k-rule is not invertible. It is of course easy to drop contraction and build it into the k-rule and into the rules in $\dot{\mathsf{X}}$, which makes all rules invertible. The reason we choose not to do this is because our cut-elimination procedure works better with explicit contraction.

There are also some rules that will turn out to be admissible, namely the $\Diamond$-rules, and the rules necessitation, weakening and cut, which are shown in Figure 2. A $\Diamond$-rule is in a certain sense the result of "reflecting" the corresponding structural rule at the cut, and vice versa. This comment will hopefully become more clear after the proof of the reduction lemma.

Given a set $\mathsf{X} \subseteq \{\mathsf{d}, \mathsf{t}, \mathsf{b}, \mathsf{4}, \mathsf{5}\}$, $\dot{\mathsf{X}}$ is the corresponding subset of $\{\dot{\mathsf{d}}, \dot{\mathsf{t}}, \dot{\mathsf{b}}, \dot{\mathsf{4}}, \dot{\mathsf{5}}\}$ and $\overset{\diamond}{\mathsf{X}}$ is the corresponding subset of $\{\overset{\diamond}{\mathsf{d}}, \overset{\diamond}{\mathsf{t}}, \overset{\diamond}{\mathsf{b}}, \overset{\diamond}{\mathsf{4}}, \overset{\diamond}{\mathsf{5}}\}$.

Inference rules, derivations, proofs. In the following instance of an inference rule ρ

$$\rho\,\frac{\Gamma_1 \quad \ldots \quad \Gamma_n}{\Delta}$$

$$\Gamma\{p,\bar{p}\} \qquad \wedge\,\frac{\Gamma\{A\} \quad \Gamma\{B\}}{\Gamma\{A \wedge B\}} \qquad \vee\,\frac{\Gamma\{A,B\}}{\Gamma\{A \vee B\}}$$

$$\Box\,\frac{\Gamma\{[A]\}}{\Gamma\{\Box A\}} \qquad \mathsf{k}\,\frac{\Gamma\{[A,\Delta]\}}{\Gamma\{\Diamond A,[\Delta]\}} \qquad \mathsf{ctr}\,\frac{\Gamma\{\Delta,\Delta\}}{\Gamma\{\Delta\}}$$

$$\dot{\mathsf{d}}\,\frac{\Gamma\{[\emptyset]\}}{\Gamma\{\emptyset\}} \qquad \dot{\mathsf{t}}\,\frac{\Gamma\{[\Delta]\}}{\Gamma\{\Delta\}} \qquad \dot{\mathsf{b}}\,\frac{\Gamma\{[\Delta,[\Sigma]]\}}{\Gamma\{[\Delta],\Sigma\}}$$

$$\dot{\mathsf{4}}\,\frac{\Gamma\{[\Delta],[\Sigma]\}}{\Gamma\{[[\Delta],\Sigma]\}} \qquad \dot{\mathsf{5}}\,\frac{\Gamma\{[\Delta]\}\{\emptyset\}}{\Gamma\{\emptyset\}\{[\Delta]\}} \quad \mathit{depth}(\Gamma\{\ \}\{\emptyset\}) > 0$$

Fig. 1. System $\mathsf{K}+\{\dot{\mathsf{d}},\dot{\mathsf{t}},\dot{\mathsf{b}},\dot{\mathsf{4}},\dot{\mathsf{5}}\}$

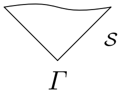
$$\overset{\diamond}{\mathsf{d}}\,\frac{\Gamma\{[A]\}}{\Gamma\{\Diamond A\}} \qquad \overset{\diamond}{\mathsf{t}}\,\frac{\Gamma\{A\}}{\Gamma\{\Diamond A\}} \qquad \overset{\diamond}{\mathsf{b}}\,\frac{\Gamma\{[\Delta],A\}}{\Gamma\{[\Delta,\Diamond A]\}}$$

$$\overset{\diamond}{\mathsf{4}}\,\frac{\Gamma\{[\Delta,\Diamond A]\}}{\Gamma\{\Diamond A,[\Delta]\}} \qquad \overset{\diamond}{\mathsf{5}}\,\frac{\Gamma\{\emptyset\}\{\Diamond A\}}{\Gamma\{\Diamond A\}\{\emptyset\}} \quad \mathit{depth}(\Gamma\{\ \}\{\emptyset\}) > 0$$

$$\mathsf{nec}\,\frac{\Gamma}{[\Gamma]} \qquad \mathsf{wk}\,\frac{\Gamma\{\emptyset\}}{\Gamma\{\Delta\}} \qquad \mathsf{cut}\,\frac{\Gamma\{A\} \qquad \Gamma\{\bar{A}\}}{\Gamma\{\emptyset\}}$$

Fig. 2. Diamond rules, necessitation, weakening, cut

we call $\Gamma_1 \ldots \Gamma_n$ its *premises* and Δ its *conclusion*. We write ρ^n to denote n instances of ρ and ρ^* to denote an unspecified number of instances of ρ. A *system*, denoted by $\mathcal{S}$, is a set of inference rules. A *derivation* in a system $\mathcal{S}$ is a finite tree whose nodes are labelled with sequents and which is built according to the inference rules from $\mathcal{S}$. The sequent at the root is the *conclusion* and the sequents at the leaves are the *premises* of the derivation. Derivations are denoted by $\mathcal{D}$. A derivation $\mathcal{D}$ with conclusion Γ in system $\mathcal{S}$ is sometimes shown as



.

The depth of a derivation $\mathcal{D}$ is denoted by $|\mathcal{D}|$. A *proof* of a sequent Γ in a system is a derivation in this system with conclusion Γ and where all premises are instances of the axiom $\Gamma\{p,\bar{p}\}$. Proofs are denoted by $\mathcal{P}$. We write $\mathcal{S} \vdash \Gamma$ if there is a proof of Γ in system $\mathcal{S}$. An inference rule ρ is *(depth-preserving)* *admissible* for a system $\mathcal{S}$ if for each proof in $\mathcal{S} \cup \{\rho\}$ there is a proof of in $\mathcal{S}$ with the same conclusion (and with at most the same depth).

Soundness of our systems is easily established similarly to soundness of the systems in [4]:

Theorem 1 (Soundness). *Let* $\mathsf{X} \subseteq \{\mathsf{d}, \mathsf{t}, \mathsf{b}, \mathsf{4}, \mathsf{5}\}$. *If a sequent is provable in* $\mathsf{K} + \dot{\mathsf{X}}$ *then its corresponding formula is provable in a Hilbert system for the modal logic* K *extended by the axioms in* X.

Our main result is cut-elimination, which we prove in the next section.

Theorem 2 (Cut-Elimination). *Let* $\mathsf{X} \subseteq \{\mathsf{d}, \mathsf{t}, \mathsf{b}, \mathsf{4}, \mathsf{5}\}$. *If* $\mathsf{K} + \dot{\mathsf{X}} + \mathsf{cut} \vdash \Gamma$ *then* $\mathsf{K} + \dot{\mathsf{X}} \vdash \Gamma$.

By using cut-elimination we obtain the completeness theorem:

Theorem 3 (Completeness). *Let* $\mathsf{X} \subseteq \{\mathsf{d}, \mathsf{t}, \mathsf{b}, \mathsf{4}, \mathsf{5}\}$. *If a formula is provable in a Hilbert system for the modal logic* K *extended by the modal axioms in* X *then it is provable in system* $\mathsf{K} + \dot{\mathsf{X}}$.

Proof. Given a proof in the Hilbert system we construct a proof in $\mathsf{K} + \dot{\mathsf{X}} + \mathsf{cut}$ as usual, and then apply Theorem 2 (Cut-elimination). We show proofs for the modal axioms:

$$\vee\,\dfrac{\dot{\mathsf{d}}\,\dfrac{\mathsf{k}^2\,\dfrac{[\bar{A}, A]}{\Diamond\bar{A}, \Diamond A, [\emptyset]}}{\Diamond\bar{A}, \Diamond A}}{\Box A \supset \Diamond A}
\qquad
\vee\,\dfrac{\dot{\mathsf{t}}\,\dfrac{\mathsf{k}\,\dfrac{[A, \bar{A}]}{[\bar{A}], \Diamond A}}{\bar{A}, \Diamond A}}{A \supset \Diamond A}
\qquad
\vee\,\dfrac{\Box\,\dfrac{\dot{\mathsf{b}}\,\dfrac{\mathsf{k}\,\dfrac{[[A, \bar{A}]]}{[[\bar{A}], \Diamond A]}}{\bar{A}, [\Diamond A]}}{\bar{A}, \Box\Diamond A}}{A \supset \Box\Diamond A}$$

$$\vee\,\dfrac{\Box^2\,\dfrac{\dot{\mathsf{4}}\,\dfrac{\mathsf{k}\,\dfrac{[\bar{A}, A], [\emptyset]}{\Diamond\bar{A}, [A], [\emptyset]}}{\Diamond\bar{A}, [[A]]}}{\Diamond\bar{A}, \Box\Box A}}{\Box A \supset \Box\Box A}
\qquad
\vee\,\dfrac{\Box^2\,\dfrac{\dot{\mathsf{5}}\,\dfrac{\mathsf{k}\,\dfrac{[[\bar{A}, A]]}{[[\bar{A}], \Diamond A]}}{[\bar{A}], [\Diamond A]}}{\Box\bar{A}, \Box\Diamond A}}{\Diamond A \supset \Box\Diamond A}$$

□

3 Syntactic Cut-Elimination

We first need some definitions. The *depth* of a formula A, denoted $\mathit{depth}(A)$, is defined as usual, the depth of possibly negated atoms being zero. Given an instance of the cut rule as shown in Figure 2, its *cut formula* is A and its *cut rank* is one plus the depth of its cut formula. For $r \geq 0$ we define the rule cut_r which is cut with at most rank r. The *cut rank* of a derivation is the supremum of the cut ranks of its cuts. A rule is *cut-rank (and depth-) preserving admissible* for a system $\mathcal{S}$ if for all $r \geq 0$ the rule is (depth-preserving) admissible for $\mathcal{S} + \mathsf{cut}_r$.

Lemma 1 (Weakening and necessitation admissibility). *Let* $\mathsf{X} \subseteq \{\mathsf{d}, \mathsf{t}, \mathsf{b}, \mathsf{4}, \mathsf{5}\}$. *The* wk*-rule and the* nec*-rule are depth- and cut-rank-preserving admissible for* $\mathsf{K} + \dot{\mathsf{X}}$.

Proof. A routine induction shows that a single nec or wk-rule can be eliminated from a given proof, a second induction on the number of nec or wk-rules yields our lemma. □

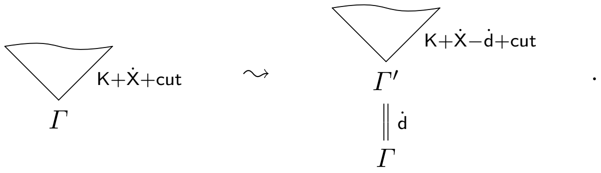
$$\mathsf{m}\Box\,\frac{\Gamma\{[A,\ldots,A]\}}{\Gamma\{\Box A\}} \qquad \mathsf{m}\wedge\,\frac{\Gamma\{A,\ldots,A\}\quad \Gamma\{B,\ldots,B]\}}{\Gamma\{A\wedge B\}}$$

$$\mathsf{mcut}\,\frac{\Gamma\{A,\ldots,A\}\quad \Gamma\{\bar{A},\ldots,\bar{A}\}}{\Gamma\{\emptyset\}} \qquad \mathsf{med}\,\frac{\Gamma\{[\Delta],[\Sigma]\}}{\Gamma\{[\Delta,\Sigma]\}} \qquad \mathsf{fctr}\,\frac{\Gamma\{A,A\}}{\Gamma\{A\}}$$

Fig. 3. Multi-rules, medial, and formula contraction

Seriality (the rule $\dot{\mathsf{d}}$) is different from the other rules: it trivially permutes below the cut. So we can get it out of the way and then prove cut elimination for the systems without seriality.

Lemma 2 (Push down seriality). *Let* $\mathsf{X} \subseteq \{\mathsf{d},\mathsf{t},\mathsf{b},\mathsf{4},\mathsf{5}\}$ *and* $\mathsf{d}\in\mathsf{X}$. *For each proof as shown on the left there is a proof as shown on the right:*



Proof. By an easy permutation argument, making use of weakening admissibility.

We also get contraction out of the way in order to eliminate the cut. First, we decompose contraction into the fctr-rule, which is contraction on formulas, and the med-rule, shown in Figure 3. We permute down the fctr-rule. It does not permute down below the rules cut, $\Box$ and $\wedge$, so we generalise these rules as in Figure 3. We define a contraction-free system K^- as $\mathsf{K}^- = \mathsf{K} - \mathsf{ctr} + \{\mathsf{med}, \mathsf{m}\Box, \mathsf{m}\wedge\}$ and will show cut elimination for that system, but first we develop the machinery to show that cut elimination for K^- leads to cut-elimination for K (with any $\dot{\mathsf{X}}$).

Lemma 3 (Decompose contraction). *The* ctr*-rule is derivable for* $\{\mathsf{fctr},\mathsf{med}\}$.

Proof. By induction the depth of a sequent which is contracted, we show the inductive step:

$$\mathsf{ctr}\,\frac{\Gamma\{A_1,\ldots,A_m,[\Delta_1],\ldots,[\Delta_n],A_1,\ldots,A_m,[\Delta_1],\ldots,[\Delta_n]\}}{\Gamma\{A_1,\ldots,A_m,[\Delta_1],\ldots,[\Delta_n]\}}$$

$$\rightsquigarrow\qquad \mathsf{fctr}^m\,\cfrac{\mathsf{ctr}^n\,\cfrac{\mathsf{med}^n\,\cfrac{\Gamma\{A_1,\ldots,A_m,[\Delta_1],\ldots,[\Delta_n],A_1,\ldots,A_m,[\Delta_1],\ldots,[\Delta_n]\}}{\Gamma\{A_1,\ldots,A_m,A_1,\ldots,A_m,[\Delta_1,\Delta_1],\ldots,[\Delta_n,\Delta_n]\}}}{\Gamma\{A_1,\ldots,A_m,A_1,\ldots,A_m,[\Delta_1],\ldots,[\Delta_n]\}}}{\Gamma\{A_1,\ldots,A_m,[\Delta_1],\ldots,[\Delta_n]\}}$$

□

Lemma 4 (Weakening and necessitation admissibility for K^-). *Let* $\mathsf{X} \subseteq \{\mathsf{d}, \mathsf{t}, \mathsf{b}, 4, 5\}$. *The* wk*-rule and the* nec*-rule are depth- and cut-rank-preserving admissible for* $\mathsf{K}^- + \dot{\mathsf{X}}$.

Lemma 5 (From mcut to cut). *The rule* mcut_r *is derivable for* $\{\mathsf{cut}_r, \mathsf{wk}\}$.

Proof. We define the rule $\mathsf{mcut}_r^{m,n}$ with $m, n > 0$ as

$$\frac{\Gamma\{\overbrace{A,\dots,A}^{m-\text{times}}\} \quad \Gamma\{\overbrace{\bar{A},\dots,\bar{A}}^{n-\text{times}}\}}{\Gamma\{\emptyset\}} \qquad ,$$

and show that rule derivable for $\{\mathsf{cut}_r, \mathsf{wk}\}$ by induction on $m + n$. The case for $m = n = 1$ is trivial, for $m > 1$ and $n = 1$ we replace

$$\mathsf{mcut}_r^{m,1}\,\frac{\Gamma\{A,\dots,A\} \quad \Gamma\{\bar{A}\}}{\Gamma\{\emptyset\}}$$

by

$$\mathsf{cut}_r\,\frac{\mathsf{mcut}_r^{m-1,1}\,\dfrac{\Gamma\{A,\dots,A\} \quad \mathsf{wk}\,\dfrac{\Gamma\{\bar{A}\}}{\Gamma\{\bar{A},A\}}}{\Gamma\{A\}} \qquad \Gamma\{\bar{A}\}}{\Gamma\{\emptyset\}}$$

and apply the induction hypothesis, and for $m, n > 1$ we replace

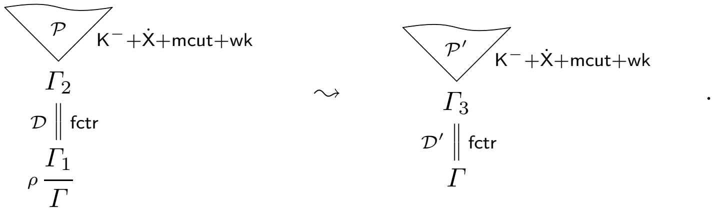
$$\mathsf{mcut}_r^{m,n}\,\frac{\Gamma\{A,\dots,A\} \quad \Gamma\{\bar{A},\dots,\bar{A}\}}{\Gamma\{\emptyset\}}$$

by

$$\mathsf{cut}_r\,\frac{\mathsf{mcut}_r^{m-1,n}\,\dfrac{\Gamma\{A,\dots,A\} \quad \mathsf{wk}\,\dfrac{\Gamma\{\bar{A},\dots,\bar{A}\}}{\Gamma\{\bar{A},\dots,\bar{A},A\}}}{\Gamma\{A\}} \qquad \mathsf{mcut}_r^{m,n-1}\,\dfrac{\mathsf{wk}\,\dfrac{\Gamma\{A,\dots,A\}}{\Gamma\{A,\dots,A,\bar{A}\}} \quad \Gamma\{\bar{A},\dots,\bar{A}\}}{\Gamma\{\bar{A}\}}}{\Gamma\{\emptyset\}}$$

and apply the induction hypothesis twice. □

Lemma 6 (Push down contraction). *Let* $\mathsf{X} \subseteq \{\mathsf{t}, \mathsf{b}, 4, 5\}$. *Given a proof as shown on the left, with* ρ *a single-premise-rule from* $\mathsf{K}^- + \dot{\mathsf{X}} + \mathsf{wk}$, *there is a proof as shown on the right, with* $|\mathcal{D}'| \leq |\mathcal{D}|$:



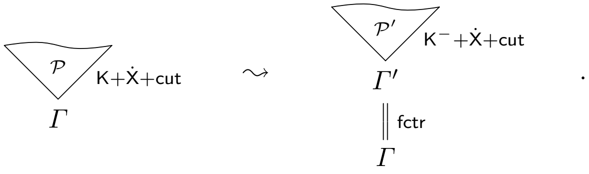
Proof. By induction on the length of $\mathcal{D}$ and a case analysis on ρ. Most cases are trivial. We show the two interesting ones. For $\rho = \vee$ and $\rho = \mathsf{k}$ we apply the following transformations:

$$\vee\,\dfrac{\mathsf{fctr}\,\dfrac{\Gamma\{A,A,B\}}{\Gamma\{A,B\}}}{\Gamma\{A\vee B\}} \quad\rightsquigarrow\quad \mathsf{fctr}\,\dfrac{\vee^2\,\dfrac{\mathsf{wk}\,\dfrac{\Gamma\{A,A,B\}}{\Gamma\{A,B,A,B\}}}{\Gamma\{A\vee B,A\vee B\}}}{\Gamma\{A\vee B\}}$$

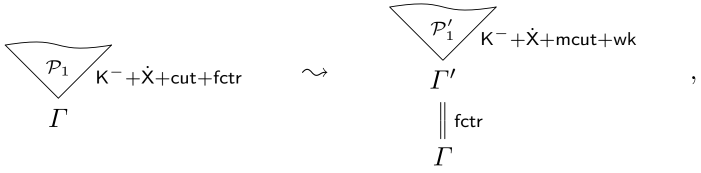
$$\mathsf{k}\,\dfrac{\mathsf{fctr}\,\dfrac{\Gamma\{[A,A,\Delta]\}}{\Gamma\{[A,\Delta]\}}}{\Gamma\{\diamond A,[\Delta]\}} \quad\rightsquigarrow\quad \mathsf{fctr}\,\dfrac{\mathsf{k}^2\,\dfrac{\Gamma\{[A,A,\Delta]\}}{\Gamma\{\diamond A,\diamond A,[\Delta]\}}}{\Gamma\{\diamond A,[\Delta]\}} \quad ,$$

and in each case we apply the induction hypothesis twice. □

Proposition 1 (Push down contraction). *Given a proof as shown on the left, there is a proof as shown on the right:*



Proof. We first prove the claim that for each proof as shown on the left there is a proof as shown on the right:



The proof of the claim is by induction on the depth of $\mathcal{P}_1$, using Lemma 6 (Push down contraction). The proof of our proposition is as follows: by Lemma 3 (Decompose contraction) we obtain a proof in $\mathsf{K}^- + \dot{\mathsf{X}} + \mathsf{cut} + \mathsf{fctr}$, we apply our claim, then we use Lemma 5 (From mcut to cut), to replace mcut, starting with the top-most instances. Finally we remove weakening using weakening admissibility. □

The following three lemmas are needed for the reduction lemma. We define

$$\mathsf{X}^+ = \begin{cases} \mathsf{X}\cup\{4\} & \text{if } \{\mathsf{t},5\}\subseteq\mathsf{X} \text{ or } \{\mathsf{b},5\}\subseteq\mathsf{X} \\ \mathsf{X}\cup\{5\} & \text{if } \{\mathsf{b},4\}\subseteq\mathsf{X} \\ \mathsf{X} & \text{otherwise} \quad , \end{cases}$$

and likewise for $\overset{\diamond}{\mathsf{X}}$ and $\dot{\mathsf{X}}$.

Lemma 7 (Push down 45). *Let* $\mathsf{X} \subseteq \{\mathsf{t},\mathsf{b},\mathsf{4},\mathsf{5}\}$ *and* $\rho \in (\overset{\diamond}{\mathsf{X}} \cap \{\overset{\diamond}{\mathsf{4}},\overset{\diamond}{\mathsf{5}}\})$. *Given a derivation as shown on the left, where* ρ *applies to* $\Diamond A$*, there is a derivation as shown on the right, where all rules in* $\mathcal{D}_3$ *apply to the instance of* $\Diamond A$ *shown, and where* $|\mathcal{D}_2| \leq |\mathcal{D}_1|$*:*

$$\begin{array}{c}\rho\,\dfrac{\Gamma\{\Diamond A\}}{\Gamma_1\{\Diamond A\}}\\ \mathcal{D}_1 \Big\Vert\, \dot{\mathsf{X}}+\mathsf{med} \\ \Delta\{\Diamond A\}\end{array} \quad\rightsquigarrow\quad \begin{array}{c} \Gamma\{\Diamond A\}\\ \mathcal{D}_2 \Big\Vert\, \dot{\mathsf{X}}+\mathsf{med}\\ \Gamma_2\{\Diamond A\}\\ \mathcal{D}_3\Big\Vert\,(\overset{\diamond}{\mathsf{X}}{}^{+}\cap\{\overset{\diamond}{\mathsf{4}},\overset{\diamond}{\mathsf{5}}\})\\ \Delta\{\Diamond A\}\end{array}\quad .$$

Proof. The proof is by induction on the length of $\mathcal{D}_1$. We permute the instance of ρ down and apply the induction hypothesis, possibly several times. We only show the non-trivial permutations.

$$\mathsf{med}\,\dfrac{\overset{\diamond}{\mathsf{4}}\,\dfrac{\Gamma\{[\Diamond A,\Delta],[\Sigma]\}}{\Gamma\{\Diamond A,[\Delta],[\Sigma]\}}}{\Gamma\{\Diamond A,[\Delta,\Sigma]\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{4}}\,\dfrac{\mathsf{med}\,\dfrac{\Gamma\{[\Diamond A,\Delta],[\Sigma]\}}{\Gamma\{[\Diamond A,\Delta,\Sigma]\}}}{\Gamma\{\Diamond A,[\Delta,\Sigma]\}}$$

$$\dot{\mathsf{t}}\,\dfrac{\overset{\diamond}{\mathsf{4}}\,\dfrac{\Gamma\{[\Diamond A,\Delta]\}}{\Gamma\{\Diamond A,[\Delta]\}}}{\Gamma\{\Diamond A,\Delta\}} \quad\rightsquigarrow\quad \dot{\mathsf{t}}\,\dfrac{\Gamma\{[\Diamond A,\Delta]\}}{\Gamma\{\Diamond A,\Delta\}}$$

$$\dot{\mathsf{b}}\,\dfrac{\overset{\diamond}{\mathsf{4}}\,\dfrac{\Gamma\{[\Delta,[\Diamond A,\Sigma]]\}}{\Gamma\{[\Diamond A,\Delta,[\Sigma]]\}}}{\Gamma\{[\Diamond A,\Delta],\Sigma\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{5}}\,\dfrac{\dot{\mathsf{b}}\,\dfrac{\Gamma\{[\Delta,[\Diamond A,\Sigma]]\}}{\Gamma\{\Diamond A,[\Delta],\Sigma\}}}{\Gamma\{[\Diamond A,\Delta],\Sigma\}}$$

$$\dot{\mathsf{4}}\,\dfrac{\overset{\diamond}{\mathsf{4}}\,\dfrac{\Gamma\{[\Diamond A,\Delta],[\Sigma]\}}{\Gamma\{\Diamond A,[\Delta],[\Sigma]\}}}{\Gamma\{\Diamond A,[[\Delta],\Sigma]\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{4}}\,\dfrac{\overset{\diamond}{\mathsf{4}}\,\dfrac{\dot{\mathsf{4}}\,\dfrac{\Gamma\{[\Diamond A,\Delta],[\Sigma]\}}{\Gamma\{[[\Diamond A,\Delta],\Sigma]\}}}{\Gamma\{[\Diamond A,[\Delta],\Sigma]\}}}{\Gamma\{\Diamond A,[[\Delta],\Sigma]\}}$$

$$\dot{\mathsf{5}}\,\dfrac{\overset{\diamond}{\mathsf{4}}\,\dfrac{\Gamma\{[\Diamond A,\Delta]\}\{\emptyset\}}{\Gamma\{\Diamond A,[\Delta]\}\{\emptyset\}}}{\Gamma\{\Diamond A\}\{[\Delta]\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{5}}\,\dfrac{\dot{\mathsf{5}}\,\dfrac{\Gamma\{[\Diamond A,\Delta]\}\{\emptyset\}}{\Gamma\{\emptyset\}\{[\Diamond A,\Delta]\}}}{\Gamma\{\Diamond A\}\{[\Delta]\}}$$

Permuting down the $\overset{\diamond}{\mathsf{5}}$-rule is trivial except over the $\dot{\mathsf{t}}$-rule and the $\dot{\mathsf{b}}$-rule, and this is also trivial unless the restriction on the depth of the context in the $\overset{\diamond}{\mathsf{5}}$-rule becomes relevant:

$$\dot{\mathsf{t}}\,\dfrac{\overset{\diamond}{\mathsf{5}}\,\dfrac{\Gamma_1,[\Delta],\Gamma_2\{\Diamond A\}}{\Gamma_1,[\Diamond A,\Delta],\Gamma_2\{\emptyset\}}}{\Gamma_1,\Diamond A,\Delta,\Gamma_2\{\emptyset\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{4}}{}^{*}\,\dfrac{\dot{\mathsf{t}}\,\dfrac{\Gamma_1,[\Delta],\Gamma_2\{\Diamond A\}}{\Gamma_1,\Delta,\Gamma_2\{\Diamond A\}}}{\Gamma_1,\Diamond A,\Delta,\Gamma_2\{\emptyset\}}$$

$$\dot{\mathsf{b}}\,\dfrac{\overset{\diamond}{\mathsf{5}}\,\dfrac{[\Delta,[\Sigma]],\Gamma\{\Diamond A\}}{[\Delta,[\Sigma,\Diamond A]],\Gamma\{\emptyset\}}}{[\Delta],\Sigma,\Diamond A,\Gamma\{\emptyset\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{4}}{}^{*}\,\dfrac{\dot{\mathsf{b}}\,\dfrac{[\Delta,[\Sigma]],\Gamma\{\Diamond A\}}{[\Delta],\Sigma,\Gamma\{\Diamond A\}}}{[\Delta],\Sigma,\Diamond A,\Gamma\{\emptyset\}}$$

□

Lemma 8 (Push down ktb). *Let $\mathsf{X}\subseteq\{\mathsf{t},\mathsf{b},\mathsf{4},\mathsf{5}\}$ and let $\rho=\mathsf{k}$ or $\rho\in(\overset{\diamond}{\mathsf{X}}\cap\{\overset{\diamond}{\mathsf{t}},\overset{\diamond}{\mathsf{b}}\})$. Given a derivation as shown on the left, where ρ applies to $\Diamond A$, there is a derivation as shown on the right, with $\sigma=\mathsf{k}$ or $\sigma\in(\overset{\diamond}{\mathsf{X}}\cap\{\overset{\diamond}{\mathsf{t}},\overset{\diamond}{\mathsf{b}}\})$, where all rules in $\mathcal{D}_3$ apply to the instance of $\Diamond A$ shown, and where $|\mathcal{D}_2|\le|\mathcal{D}_1|$:*

$$\begin{array}{c} \rho\,\dfrac{\Gamma\{A\}}{\Gamma_1\{\Diamond A\}} \\ \mathcal{D}_1 \Big\Vert\; \dot{\mathsf{X}}{+}\mathsf{med} \\ \Delta\{\Diamond A\} \end{array} \quad\rightsquigarrow\quad \begin{array}{c} \Gamma\{A\} \\ \mathcal{D}_2 \Big\Vert\; \dot{\mathsf{X}}{+}\mathsf{med} \\ \sigma\,\dfrac{\Gamma_3\{A\}}{\Gamma_2\{\Diamond A\}} \\ \mathcal{D}_3 \Big\Vert\; (\overset{\diamond}{\mathsf{X}}{}^{+}\cap\{\overset{\diamond}{\mathsf{4}},\overset{\diamond}{\mathsf{5}}\}) \\ \Delta\{\Diamond A\} \end{array} \quad .$$

Proof. The proof is by induction on the length of $\mathcal{D}_1$. We permute the instance of ρ down and apply Lemma 7 (Push down $\mathsf{45}$) and/or the induction hypothesis. We only show the non-trivial permutations.

$$\dot{\mathsf{t}}\,\dfrac{\mathsf{k}\,\dfrac{\Gamma\{[A,\Delta]\}}{\Gamma\{\Diamond A,[\Delta]\}}}{\Gamma\{\Diamond A,\Delta\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{t}}\,\dfrac{\dot{\mathsf{t}}\,\dfrac{\Gamma\{[A,\Delta]\}}{\Gamma\{A,\Delta\}}}{\Gamma\{\Diamond A,\Delta\}}$$

$$\dot{\mathsf{b}}\,\dfrac{\mathsf{k}\,\dfrac{\Gamma\{[\Delta,[A,\Sigma]]\}}{\Gamma\{[\Diamond A,\Delta,[\Sigma]]\}}}{\Gamma\{[\Diamond A,\Delta],\Sigma\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{b}}\,\dfrac{\dot{\mathsf{b}}\,\dfrac{\Gamma\{[\Delta,[A,\Sigma]]\}}{\Gamma\{A,[\Delta],\Sigma\}}}{\Gamma\{[\Diamond A,\Delta],\Sigma\}}$$

$$\dot{\mathsf{4}}\,\dfrac{\mathsf{k}\,\dfrac{\Gamma\{[A,\Delta],[\Sigma]\}}{\Gamma\{\Diamond A,[\Delta],[\Sigma]\}}}{\Gamma\{\Diamond A,[[\Delta],\Sigma]\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{4}}\,\dfrac{\mathsf{k}\,\dfrac{\dot{\mathsf{4}}\,\dfrac{\Gamma\{[A,\Delta],[\Sigma]\}}{\Gamma\{[[A,\Delta],\Sigma]\}}}{\Gamma\{[\Diamond A,[\Delta],\Sigma]\}}}{\Gamma\{\Diamond A,[[\Delta],\Sigma]\}}$$

$$\dot{\mathsf{5}}\,\dfrac{\mathsf{k}\,\dfrac{\Gamma\{[A,\Delta]\}\{\emptyset\}}{\Gamma\{\Diamond A,[\Delta]\}\{\emptyset\}}}{\Gamma\{\Diamond A\}\{[\Delta]\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{5}}\,\dfrac{\mathsf{k}\,\dfrac{\dot{\mathsf{5}}\,\dfrac{\Gamma\{[A,\Delta]\}\{\emptyset\}}{\Gamma\{\emptyset\}\{[A,\Delta]\}}}{\Gamma\{\emptyset\}\{\Diamond A,[\Delta]\}}}{\Gamma\{\Diamond A\}\{[\Delta]\}}$$

The cases for $\rho=\overset{\diamond}{\mathsf{t}}$ are trivial.

$$\dot{\mathsf{t}}\,\dfrac{\overset{\diamond}{\mathsf{b}}\,\dfrac{\Gamma\{[\Delta],A\}}{\Gamma\{[\Delta,\Diamond A]\}}}{\Gamma\{\Delta,\Diamond A\}} \quad\rightsquigarrow\quad \overset{\diamond}{\mathsf{t}}\,\dfrac{\dot{\mathsf{t}}\,\dfrac{\Gamma\{[\Delta,A]\}}{\Gamma\{\Delta,A\}}}{\Gamma\{\Delta,\Diamond A\}}$$

$$\dot{\mathsf{b}}\,\dfrac{\overset{\diamond}{\mathsf{b}}\,\dfrac{\Gamma\{[\Sigma,[\Delta],A]\}}{\Gamma\{[\Sigma,[\Delta,\Diamond A]]\}}}{\Gamma\{[\Sigma],\Delta,\Diamond A\}} \quad\leadsto\quad \mathsf{k}\,\dfrac{\dot{\mathsf{b}}\,\dfrac{\Gamma\{[\Sigma,[\Delta],A]\}}{\Gamma\{[\Sigma,A],\Delta\}}}{\Gamma\{[\Sigma],\Delta,\Diamond A\}}$$

$$\dot{\mathsf{4}}\,\dfrac{\overset{\diamond}{\mathsf{b}}\,\dfrac{\Gamma\{[\Delta],A,[\Sigma]\}}{\Gamma\{[\Delta,\Diamond A],[\Sigma]\}}}{\Gamma\{[[\Delta,\Diamond A],\Sigma]\}} \quad\leadsto\quad \overset{\diamond}{\mathsf{5}}\,\dfrac{\overset{\diamond}{\mathsf{b}}\,\dfrac{\dot{\mathsf{4}}\,\dfrac{\Gamma\{[\Delta],A,[\Sigma]\}}{\Gamma\{[[\Delta],\Sigma],A\}}}{\Gamma\{[[\Delta],\Diamond A,\Sigma]\}}}{\Gamma\{[[\Delta,\Diamond A],\Sigma]\}}$$

For permuting down over the $\dot{\mathsf{5}}$-rule, in the only non-trivial case, notice that the context has to be of the form shown because of the restriction of context depth in the $\dot{\mathsf{5}}$-rule:

$$\dot{\mathsf{5}}\,\dfrac{\overset{\diamond}{\mathsf{b}}\,\dfrac{\Gamma\{\emptyset\}\{[\Sigma,[\Delta],A]\}}{\Gamma\{\emptyset\}\{[\Sigma,[\Delta,\Diamond A]]\}}}{\Gamma\{[\Delta,\Diamond A]\}\{[\Sigma,\emptyset]\}} \quad\leadsto\quad \overset{\diamond}{\mathsf{5}}\,\dfrac{\mathsf{k}\,\dfrac{\dot{\mathsf{5}}\,\dfrac{\Gamma\{\emptyset\}\{[\Sigma,[\Delta],A]\}}{\Gamma\{[\Delta]\}\{[A,\Sigma]\}}}{\Gamma\{[\Delta]\}\{\Diamond A,[\Sigma,\emptyset]\}}}{\Gamma\{[\Delta,\Diamond A]\}\{[\Sigma,\emptyset]\}}$$

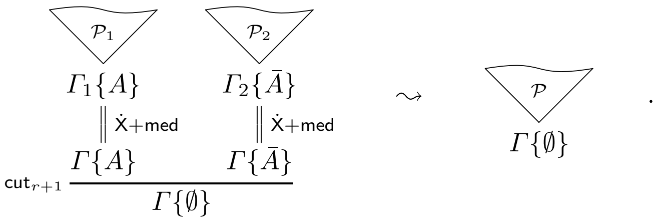
□

Lemma 9 (Reflect $\mathsf{45}$). *Let* $\mathsf{X} \subseteq \{\mathsf{4},\mathsf{5}\}$. *Given a derivation as shown on the left, where all rules in* $\mathcal{D}$ *apply to the instance of* $\Diamond A$ *shown, then for each sequent* Δ *there is a derivation as shown on the right:*

$$\begin{array}{c} \Gamma\{\Diamond A\}\{\emptyset\} \\ \mathcal{D}\,\Big\Vert\,\overset{\diamond}{\mathsf{X}} \\ \Gamma\{\emptyset\}\{\Diamond A\} \end{array} \quad\leadsto\quad \begin{array}{c} \Gamma\{\emptyset\}\{[\Delta]\} \\ \mathcal{D}'\,\Big\Vert\,\dot{\mathsf{X}} \\ \Gamma\{[\Delta]\}\{\emptyset\} \end{array} \quad .$$

Proof. By induction on the length of $\mathcal{D}$. □

Lemma 10 (Reduction Lemma). *Let* $\mathsf{X} \subseteq \{\mathsf{t},\mathsf{b},\mathsf{4},\mathsf{5}\}$. *Given a proof as shown on the left, with* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *in* $\mathsf{K}^-{+}\dot{\mathsf{X}}{+}\mathsf{cut}_r$, *then there is a proof* $\mathcal{P}$ *in* $\mathsf{K}^-{+}\dot{\mathsf{X}}^+{+}\mathsf{cut}_r$ *as shown on the right:*



Proof. As usual, by an induction on $|\mathcal{P}_1| + |\mathcal{P}_2|$ and a case analysis on the lowermost rules in $\mathcal{P}_1$ and $\mathcal{P}_2$. We only show the most complicated case, in which we cut a box introduced by the $\mathsf{m}\Box$-rule against a diamond introduced by k-rule. All other cases are much simpler. We have

$$\mathsf{cut}_{r+1}\,\dfrac{\begin{array}{c}\mathsf{m}\Box\,\dfrac{\Gamma_1\{[B,\ldots,B]\}}{\Gamma_1\{\Box B\}}\\ \Big\Vert\ \dot{\mathsf{X}}{+}\mathsf{med}\\ \Gamma\{\Box B\}\end{array}\qquad\begin{array}{c}\mathsf{k}\,\dfrac{\Gamma_2'\{[\bar{B},\Delta]\}}{\Gamma_2'\{\Diamond\bar{B},[\Delta]\}}\\ \Big\Vert\ \dot{\mathsf{X}}{+}\mathsf{med}\\ \Gamma\{\Diamond\bar{B}\}\end{array}}{\Gamma\{\emptyset\}}$$

In the left subderivation we permute down the instance of $\mathsf{m}\Box$ and on the right subderivation we apply Lemma 8 (Push ktb down) in order to obtain the following derivation, where $\Gamma\{\ \} = \Gamma\{\ \}\{\emptyset\}$. Note that the second hole in the binary context marks the position to which the $\Diamond\bar{B}$ is moved:

$$\mathsf{cut}_{r+1}\,\dfrac{\begin{array}{c}\Gamma_1\{[B,\ldots,B]\}\\ \Big\Vert\ \dot{\mathsf{X}}{+}\mathsf{med}\\ \mathsf{m}\Box\,\dfrac{\Gamma\{[B,\ldots,B]\}\{\emptyset\}}{\Gamma\{\Box B\}\{\emptyset\}}\end{array}\qquad\begin{array}{c}\Gamma_2'\{[\bar{B},\Delta]\}\\ \Big\Vert\ \dot{\mathsf{X}}{+}\mathsf{med}\\ \sigma\,\dfrac{\Gamma_3\{\bar{B}\}}{\Gamma\{\emptyset\}\{\Diamond\bar{B}\}}\\ \Big\Vert\ (\mathsf{X}^{+}\cap\{4,5\})^{\Diamond}\\ \Gamma\{\Diamond\bar{B}\}\{\emptyset\}\end{array}}{\Gamma\{\emptyset\}\{\emptyset\}}$$

By using Lemma 9 (Reflect $\mathsf{45}$) we obtain a derivation $\mathcal{D}$ and build:

$$\mathsf{cut}_{r+1}\,\dfrac{\begin{array}{c}\Gamma_1\{[B,\ldots,B]\}\\ \Big\Vert\ \dot{\mathsf{X}}{+}\mathsf{med}\\ \Gamma\{[B,\ldots,B]\}\{\emptyset\}\\ \mathcal{D}\ \Big\Vert\ (\mathsf{X}^{+}\cap\{4,5\})^{\cdot}\\ \mathsf{m}\Box\,\dfrac{\Gamma\{\emptyset\}\{[B,\ldots,B]\}}{\Gamma\{\emptyset\}\{\Box B\}}\end{array}\qquad\begin{array}{c}\Gamma_2'\{[\bar{B},\Delta]\}\\ \Big\Vert\ \dot{\mathsf{X}}{+}\mathsf{med}\\ \sigma\,\dfrac{\Gamma_3\{\bar{B}\}}{\Gamma\{\emptyset\}\{\Diamond\bar{B}\}}\end{array}}{\Gamma\{\emptyset\}\{\emptyset\}}\quad .$$

We now consider the three possible cases for $\sigma \in \{\mathsf{k}, \overset{\diamond}{\mathsf{t}}, \overset{\diamond}{\mathsf{b}}\}$ and apply one of the following transformations to the relevant part of the proof:

$$\mathsf{cut}_{r+1}\,\dfrac{\mathsf{m}\Box\,\dfrac{\Sigma\{[B,\ldots,B],[\Delta]\}}{\Sigma\{\Box B,[\Delta]\}}\qquad \mathsf{k}\,\dfrac{\Sigma\{[\bar{B},\Delta]\}}{\Sigma\{\Diamond\bar{B},[\Delta]\}}}{\Sigma\{[\Delta]\}}\quad\leadsto\quad \mathsf{mcut}_r\,\dfrac{\mathsf{med}\,\dfrac{\Sigma\{[B,\ldots,B],[\Delta]\}}{\Sigma\{[B,\ldots,B,\Delta]\}}\qquad \Sigma\{[\bar{B},\Delta]\}}{\Sigma\{[\Delta]\}}$$

$$\mathsf{cut}_{r+1}\,\dfrac{\mathsf{m}\Box\,\dfrac{\Sigma\{[B,\ldots,B]\}}{\Sigma\{\Box B\}}\qquad \overset{\diamond}{\mathsf{t}}\,\dfrac{\Sigma\{\bar{B}\}}{\Sigma\{\Diamond\bar{B}\}}}{\Sigma\{\emptyset\}}\quad\leadsto\quad \mathsf{mcut}_r\,\dfrac{\dot{\mathsf{t}}\,\dfrac{\Sigma\{[B,\ldots,B]\}}{\Sigma\{B,\ldots,B\}}\qquad \Sigma\{\bar{B}\}}{\Sigma\{\emptyset\}}$$

$$\mathsf{cut}_{r+1}\,\dfrac{\mathsf{m}\Box\,\dfrac{\varSigma\{[[B,\ldots,B],\varDelta]\}}{\varSigma\{[\Box B,\varDelta]\}} \qquad \overset{\diamond}{\mathsf{b}}\,\dfrac{\varSigma\{\bar{B},[\varDelta]\}}{\varSigma\{[\Diamond\bar{B},\varDelta]\}}}{\varSigma\{[\varDelta]\}} \quad \rightsquigarrow \quad \mathsf{mcut}_{r}\,\dfrac{\dot{\mathsf{b}}\,\dfrac{\varSigma\{[[B,\ldots,B],\varDelta]\}}{\varSigma\{B,\ldots,B,\varDelta]\}} \qquad \varSigma\{\bar{B},[\varDelta]\}}{\varSigma\{[\varDelta]\}}.$$

We then eliminate mcut by using Lemma 5 (From mcut to cut) and weakening admissibility. □

Proposition 2 (Cut-elimination for K^-). *Let* $\mathsf{X} \subseteq \{\mathsf{t},\mathsf{b},\mathsf{4},\mathsf{5}\}$*. If* $\mathsf{K}^- + \dot{\mathsf{X}} + \mathsf{cut} \vdash \varGamma$ *then* $\mathsf{K}^- + \dot{\mathsf{X}}^+ \vdash \varGamma$*.*

Proof. We first prove the claim: If $\mathsf{K}^- + \dot{\mathsf{X}} + \mathsf{cut}_{r+1} \vdash \varGamma$ then $\mathsf{K}^- + \dot{\mathsf{X}}^+ + \mathsf{cut}_r \vdash \varGamma$. The claim is proved by induction on the depth of the given proof, using the reduction lemma. Our proposition then follows from an induction on the cut rank of the given proof, using the claim. □

Lemma 11 (From X^+ to X).
(i) The $\dot{\mathsf{4}}$*-rule is derivable for* $\{\dot{\mathsf{t}},\dot{\mathsf{5}},\mathsf{nec}\}$*.*
(ii) The $\dot{\mathsf{4}}$*-rule is derivable for* $\{\dot{\mathsf{b}},\dot{\mathsf{5}},\mathsf{nec}\}$*.*
(iii) The $\dot{\mathsf{5}}$*-rule is derivable for* $\{\dot{\mathsf{b}},\dot{\mathsf{4}},\mathsf{wk}\}$*.*

Proof. For (i) notice that the $\dot{\mathsf{4}}$-rule is a special case of the $\dot{\mathsf{5}}$-rule unless $\varGamma\{\ \}$ has depth zero, and thus $\varGamma\{\ \} = \varLambda, \{\ \}$. In that case we have:

$$\dot{\mathsf{4}}\,\dfrac{\varLambda,[\varDelta],[\varSigma]}{\varLambda,[[\varDelta],\varSigma]} \qquad \rightsquigarrow \qquad \dot{\mathsf{t}}\,\dfrac{\dot{\mathsf{5}}\,\dfrac{\mathsf{nec}\,\dfrac{\varLambda,[\varDelta],[\varSigma]}{[\varLambda,[\varDelta],[\varSigma]]}}{[\varLambda,[[\varDelta],\varSigma]]}}{\varLambda,[[\varDelta],\varSigma]} \quad .$$

For (ii) we again have to consider only the case where $\varGamma\{\ \} = \varLambda, \{\ \}$:

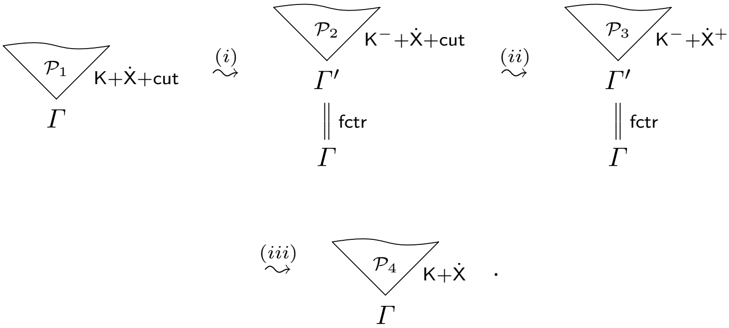
$$\dot{\mathsf{4}}\,\dfrac{\varLambda,[\varDelta],[\varSigma]}{\varLambda,[[\varDelta],\varSigma]} \qquad \rightsquigarrow \qquad \dot{\mathsf{b}}\,\dfrac{\dot{\mathsf{b}}\,\dfrac{\dot{\mathsf{5}}\,\dfrac{\mathsf{nec}^2\,\dfrac{\varLambda,[\varDelta],[\varSigma]}{[[\varLambda,[\varDelta],[\varSigma]]]}}{[[\varLambda,[\varSigma]],[\varDelta]]}}{[[\varLambda],[\varDelta],\varSigma]}}{\varLambda,[[\varDelta],\varSigma]}$$

For (iii) notice that a sequent has a tree structure and that, seen upwards, the $\dot{\mathsf{5}}$-rule allows to move a boxed sequent $[\varDelta]$ to any position in that tree, but not to the root. To move a boxed sequent to any position in the tree it is enough if we are both able to move it a) from a given node the parent of this node and b) to move it from a given node to any child of that node. Point a) is just the $\dot{\mathsf{4}}$-rule and point b) is as follows:

$$\dot{\mathsf{b}}\,\dfrac{\dot{\mathsf{4}}\,\dfrac{\mathsf{wk}\,\dfrac{\varGamma\{[\varLambda,[\varDelta]]\}}{\varGamma\{[\varLambda,[\emptyset],[\varDelta]]\}}}{\varGamma\{[\varLambda,[[\varDelta]]]\}}}{\varGamma\{[\varLambda],[\varDelta]\}} \quad .$$

□

Proof (of Theorem 2 (Cut-elimination)). We first prove the theorem for the cases where $\mathsf{d} \notin \mathsf{X}$. The transformation (i) is by Proposition 1 (Push down contraction), the transformation (ii) is Proposition 2 (Cut-elimination for K^-), and transformation (iii) is by Lemma 11 (From X^+ to X) and weakening and necessitation admissibility.


$\mathcal{P}_1$
$\mathsf{K}+\dot{\mathsf{X}}+\mathsf{cut}$
Γ
(i)
$\mathcal{P}_2$
$\mathsf{K}^-+\dot{\mathsf{X}}+\mathsf{cut}$
Γ'
fctr
Γ
(ii)
$\mathcal{P}_3$
$\mathsf{K}^-+\dot{\mathsf{X}}^+$
Γ'
fctr
Γ
(iii)
$\mathcal{P}_4$
$\mathsf{K}+\dot{\mathsf{X}}$
Γ
.

In the cases where $\mathsf{d} \in \mathsf{X}$ we first apply Lemma 2 (Push down seriality) and then proceed the same way with the upper part of the proof. □

Future work. It looks like the cut-elimination proof can be generalised. So it is our goal to devise 1) easily checkable critera on rules, which guarantee cut-elimination, and 2) a procedure which turns modal axioms into rules which satisfy these criteria. Such a generic cut-elimination procedure exists already for the display calculus, but relies on the so-called display property, and thus on the language of tense logic. Recently, such a procedure has also been proposed by Ciabattoni et al. for hypersequent systems [8]. While hypersequents do not seem to be expressive enough for the modal logics considered here, nested sequents seem to add just the right amount of extra generality to enable a similar development for modal logics.

Related work. The fact that structural rules improve modularity has been observed before by Castilho et. al. [7] in the context of tableau systems. Our ◇-rules correspond exactly to what are called propagation rules in [7]. While the focus of [7] is on giving decision procedures, our focus is on giving proof systems which support a notion of cut-elimination. This is more easily done with local rules, so in sequent systems instead of tableau systems. A further difference is that propagation rules and structural rules are mixed in [7], while here we treat systems with structural rules only (and in [3] we treated systems with propagation- or ◇-rules only).

References

1. Avron, A.: The method of hypersequents in the proof theory of propositional nonclassical logics. In: Hodges, W., Hyland, M., Steinhorn, C., Truss, J. (eds.) Logic: from foundations to applications. Proc. Logic Colloquium, Keele, UK, 1993, pp. 1–32. Oxford University Press, New York (1996)
2. Belnap Jr., N.D.: Display logic. Journal of Philosophical Logic 11, 375–417 (1982)
3. Brünnler, K.: Deep sequent systems for modal logic. In: Governatori, G., Hodkinson, I., Venema, Y. (eds.) Advances in Modal Logic, vol. 6, pp. 107–119. College Publications (2006)
4. Brünnler, K.: Deep sequent systems for modal logic. In: Archive for Mathematical Logic (to appear, 2008) http://www.iam.unibe.ch/~kai/Papers/dsm.pdf
5. Brünnler, K., Tiu, A.F.: A local system for classical logic. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS, vol. 2250, pp. 347–361. Springer, Heidelberg (2001)
6. Bull, R.A.: Cut elimination for propositional dynamic logic without *. Mathematische Logik und Grundlagen der Mathematik 38, 85–100 (1992)
7. Castilho, M.A., del Cerro, L.F., Gasquet, O., Herzig, A.: Modal tableaux with propagation rules and structural rules. Fundam. Inf. 32(3-4), 281–297 (1997)
8. Ciabattoni, A., Galatos, N., Terui, K.: From axioms to analytic rules in nonclassical logics. In: Proceedings of LICS 2008, pp. 229–240 (2008)
9. Goré, R., Tiu, A.: Classical modal display logic in the calculus of structures and minimal cut-free deep inference calculi for S5. Journal of Logic and Computation 17(4), 767–794 (2007)
10. Hein, R., Stewart, C.: Purity through unravelling. In: Bruscoli, P., Lamarche, F., Stewart, C. (eds.) Structures and Deduction, pp. 126–143. Technische Universität Dresden (2005)
11. Kashima, R.: Cut-free sequent calculi for some tense logics. Studia Logica 53, 119–135 (1994)
12. Negri, S.: Proof analysis in modal logic. Journal of Philosophical Logic 34(5-6), 507–544 (2005)
13. Poggiolesi, F.: The tree-hypersequent method for modal propositional logic. In: Malinowski, J., Makinson, D., Wansing, H. (eds.) Towards Mathematical Philosophy. Trends in Logic, pp. 9–30. Springer, Heidelberg (2009)
14. Stewart, C., Stouppa, P.: A systematic proof theory for several modal logics. In: Schmidt, R., Pratt-Hartmann, I., Reynolds, M., Wansing, H. (eds.) Advances in Modal Logic, vol. 5, pp. 309–333. King's College Publications (2005)
15. Stouppa, P.: A deep inference system for the modal logic S5. Studia Logica 85(2), 199–214 (2007)
16. Wansing, H.: Displaying Modal Logic. Trends in Logic Series, vol. 3. Kluwer Academic Publishers, Dordrecht (1998)

Abduction and Consequence Generation in a Support System for the Design of Logical Multiple-Choice Questions

Marta Cialdea Mayer

Dipartimento di Informatica e Automazione
Università di Roma Tre

Abstract. This paper presents Logitest, a system that can be used to assist the designer of multiple-choice questions aiming at verifying logical reasoning abilities (provided they can be represented in propositional logic). Beyond allowing the designer to check the status of an option with respect to the item stem (the problem presentation), i.e. whether it is derivable or consistent with the given information, the system can accomplish generative tasks (abduction and consequence generation) that can be useful both to complete the item stem, and to generate plausible distractors (incorrect options). The provably correct and complete algorithm used to perform abduction and consequence generation finds out minimal solutions with no need to compare each of them with all the others.

1 Introduction

Many educational institutions require students to take admission tests, either to check whether they have a suitable initial preparation and attitude, or to select (a limited number of) the most promising students among the applicants. Admission tests often include multiple-choice questions aiming at assessing logical reasoning abilities. The same kind of questions can be presented to students in scientific disciplines, who are generally required to be proficient in logical operations. Many logical tests have the following form: “given the description of a state of facts, which of the assertions listed below is derivable/underivable/consistent/inconsistent with the information you have?”. Often, the situation described in such questions concerns a fixed number of individuals, so that it can be represented in propositional logic.

A multiple-choice item is made up by (1) an *item stem* that presents the problem, (2) a sequence of *options*, containing (2a) the *correct option*, and (2b) several *distractor* options, i.e. incorrect alternative answers. A well designed multiple-choice item should meet some important basic requirements (see, for instance, [2,4]). Some of them concern the linguistic presentation of the question, others are of a more “logical” nature. In particular, it is required that (1) exactly only one of the options is correct, and (2) the options should be equally plausible. The first of such requirements seems quite obvious, and yet, in the case of logical test items, checking its fulfillment can often be tricky. As a matter of fact,

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 167–172, 2009.

invalid test items are not so infrequent. The second requirement is even subtler, especially for logical tests, and very often unmet.

The author of this paper experienced the difficulty in designing good logical questions when she was charged by her Faculty to edit the logical section of students' admission tests. In fact, some of the questions proposed by colleagues appeared to be incorrect, as well as others the author found in published test collections. Moreover, the designer can easily "run out of ideas". This motivated the implementation of Logitest, a tool to assist a test designer in both checking and devising multiple-choice logical test items.

2 The Main Functionalities of the System

Logitest is a simple support system for the design and check of test items of the form "given the information T, which of the following assertions is derivable/underivable/consistent/inconsistent with T?". The system is written in Objective Caml [6] and is available at Logitest web page [7].

The system takes a file in input, specifying the components of the item and the tasks to be executed. The input file contains a description of the language used in the test (predicates, object types, constants, etc.) and the item stem (the information T), represented by a set of logical formulae. The syntax used in the input file is first-order, but formulae are translated into a propositional language. In fact, the program can only deal with questions involving a finite and fixed set of objects (which are assumed to be pairwise different).

Moreover, the input file contains directives to perform the tasks described in the sequel.

Provability and consistency check. The input file can specify a set of formulae to be checked for provability from the stem T or consistency with T. Then the system checks them one by one. This is the basic task, useful to verify whether one and only one of the options represents the correct answer.

Completion of the information given in the item stem. This task is useful when the designer has run out of ideas; in this case the item stem specified in the input file contains general information and possibly some facts, that however are not sufficient to derive some given conclusion C, representing the correct option. The system generates all the minimal conjunctions of literals E_i, that are consistent with T and such that $T \cup \{E_i\} \models C$ (excluding trivial ones, i.e. such that $E_i \models C$). In other terms, the system solves the abduction problem given by T and C. A set of *abducible predicates* can also be specified, in order to obtain solutions containing only such predicates.

Generation of correct answers. The system can be asked to generate formulae which are derivable from the item stem; the logical consequences that are generated are minimal disjunctions of literals, excluding trivial ones (that are already explicitly present in the item stem). The system can similarly be asked to generate literals that are consistent/inconsistent with the stem (excluding its logical consequences).

Generation of distractors. The specification file may include a belief set, that is intended to represent possible students' *misconceptions* or an incorrect interpretation of the stem (such as, for instance, reading a logical equivalence in place of an implication). The beliefs are used to generate incorrect though "plausible" options. The system generates minimal disjunctions C of literals such that $B \models C$ and $T \not\models C$, where B is the belief set and T the item stem.

3 Techniques and Algorithms

Logical consequence and satisfiability tests are carried out by means of Ordered Binary Decision Diagrams [1]. The generation of consistent and inconsistent literals simply iterates over the set of literals and tests them one by one.

Abduction is a computationally hard problem: deciding whether a given abduction problem has a solution at all is a Σ_2^P-complete problem [5]. However, due to the rather small size of problems to be solved in the present context, it is a feasible task.

Logic-based abduction essentially consists in the generation of (non-trivial) explanations E such that $T \cup \{E\} \models F$ (where F is the "observation"), i.e. such that $T \cup \{\neg F\} \models \neg E$. Abduction and consequence generation can therefore be approached in the same way. And in fact Logitest performs the two tasks following the same methodology. From the syntactical point of view, abduction usually requires E to be a conjunction of literals. Using the same approach in consequence generation means that "interesting" consequences of the stem or the belief set are disjunctions of literals. The main difference between consequence generation and logic-based abduction is that the latter requires E to be consistent with T and such that $E \not\models F$, while interesting logical consequences of T should exclude those that are already explicitly contained in T itself. Such tests can however be performed after having generated E. Similarly, when generating distractors, i.e. consequences of the beliefs that are not implied by T, the test $T \not\models C$ is performed after the generation of C.

In the following, the symbol T^* stands for $T \cup \{\neg F\}$ in the case of an abductive task, T itself (or the belief set) in the consequence generation task. Candidates for abductive explanations are usually subjected to minimality criteria, such as subset-minimality. It is reasonable to ask the consequence generation task to fulfill the same requirement. In general, therefore, we want to find out *subset-minimal* sets S of literals such that $T^* \cup S$ is inconsistent. The algorithm used for performing both tasks takes T^* in input and returns a set S of literals. In the abduction case, the solution is the conjunction of such literals; in the consequence generation case, it is the disjunction of the complements of the literals in S.

The algorithm is inspired by [3], where a proof-theoretical abduction method based on semantic tableaux is defined. Explanations are identified on the basis of the set of open branches in a complete tableau for T^* (possibly simplified using containment), and such a characterization is the declarative counterpart of a non-deterministic algorithm that generates a single (minimal) abductive explanation, with no need to test each candidate solution against the others. However, in order to be complete, the algorithm suggested in [3] has to consider every permutation of the set of open tableaux branches. Logitest uses an improved algorithm, that iterates over a single sequence of tableau branches.

Like in [3], first of all a (simplified) set $\{F_1, ..., F_n\}$ of sets of literals representing the open branches in a complete tableaux for T^* is built (if we identify T^* with the conjunction of its elements, $\{F_1, ..., F_n\}$ represents a DNF of T^*). Then the branches are *complemented*, building the set $\gamma(T^*) = \{C_1, ..., C_n\}$ where $C_i = \{\overline{\ell} \mid \ell \in F_i\}$ ($\overline{\ell}$ being the complement of ℓ).

In order to present the algorithm, some preliminary definitions are needed. A set $S = \{\ell_1, ..., \ell_k\}$ of literals is said to *cover* a set $\Gamma = \{C_1, ..., C_n\}$, where each C_i is a set of literals, if for all $C_i \in \Gamma$, $S \cap C_i \neq \emptyset$ (i.e. S contains at least one element from each C_i).

Obviously, $T^* \cup S$ is inconsistent iff S covers $\gamma(T^*)$. Therefore, the search space of candidate solutions is constituted by the sets covering $\gamma(T^*)$. Candidate solutions S are generated by a backtracking algorithm which builds S in an incremental manner, scanning the list of elements of $\gamma(T^*)$.

Minimality is dealt with by means of the following notion. A set of literals S is said to be *admissible* with respect to a set Γ of sets of literals if and only if for all $\ell \in S$ there exists $C \in \Gamma$ such that $S \cap C = \{\ell\}$. I.e. S is not admissible wrt Γ iff there exists a literal $\ell \in S$ such that for all $C \in \Gamma$ containing ℓ, C contains also some other literal $\ell' \in S$. This means that the literal ℓ is *redundant*: if S covers Γ, ℓ can be eliminated from S obtaining a set that still covers Γ.

Finally, an *admissible covering* of Γ is a set that covers Γ and is admissible wrt Γ, and a *non-admissible covering* of Γ is a set that covers Γ and is not admissible wrt Γ.

It can easily be proved that:

P1. If S covers Γ, then it is subset-minimal (i.e. no $S' \subset S$ covers Γ) if and only if it is admissible wrt Γ.

Therefore, S is a minimal consistent set of literals such that $T^* \cup S$ is inconsistent iff S is a consistent and admissible covering of $\gamma(T^*)$.

The algorithm used by Logitest builds a consistent solution S incrementally, in such a way that it is always admissible. Initially, $S = \emptyset$. Then the elements of $\gamma(T^*)$ are considered one by one (in any fixed order). Each $C_i \in \gamma(T^*)$ such that $S \cap C_i \neq \emptyset$, is ignored. If $S \cap C_i = \emptyset$, a literal ℓ from C_i consistent with S is chosen; if either no literal in C_i is consistent with S of $S \cup \{\ell\}$ is not admissible wrt $\{C_1, ..., C_i\}$, the algorithm fails (and possibly backtracks), otherwise goes on to C_{i+1}, with $S \cup \{\ell\}$. Backtracking on every choice point generates all the minimal solutions of the problem.

Here follows the pseudo-code of the backtracking algorithm. In the description of the algorithm, **choose** identifies a backtrackable choice point.[1]

[1] The algorithm suggested by [3] builds candidate solutions in a similar way, but for the fact that the admissibility test is replaced by a restriction on literal choices: a literal ℓ cannot be added to S_i at stage i if $\ell \in C_1 \cup ... \cup C_{i-1}$. If such an algorithm is run only on a fixed permutation of $\{C_1, ..., C_n\}$, it is incomplete.

```
Input: $T^*$, a set of formulae
Output: a set of literals
 1) Build a simplified DNF $\{F_1, ..., F_n\}$ of $T^*$
 2) Build the set $\gamma(T^*) = \{C_1, ..., C_n\}$, where $C_i = \{\overline{\ell} \mid \ell \in F_i\}$
 3) Initialize $S_0 \leftarrow \emptyset$
 4) for $i = 1, ..., n$
 5) do if $S_{i-1} \cap C_i = \emptyset$ then
 6)       choose $\ell \in C_i$ such that $\overline{\ell} \notin S_{i-1}$
 7)       if there is no such $\ell$ then fail
 8)       else $S_i \leftarrow \{\ell\} \cup S_{i-1}$
 9)             if $S_i$ is not admissible wrt $\{C_1, ..., C_i\}$, then fail
10) done
11) return $S_n$
```

The algorithm is obviously correct: S_n does not contain any pair of complementary literals (test at line 7), it covers $\gamma(T^*)$ and is admissible wrt $\gamma(T^*)$ (otherwise S_n is rejected by the test at line 9 in the last iteration). Hence, by P1, it outputs only minimal consistent sets S such that $T^* \cup S$ is inconsistent.

With respect to completeness, let's note that the test at line 7 rejects inconsistent solutions and the test at line 5 rules out solutions that would trivially be non minimal. However, in principle, the algorithm could be incomplete, if a set S_i is discarded because it is not admissible wrt $\{C_1, ..., C_i\}$, but it is admissible wrt $\{C_1, ..., C_{i+1}\}$ (or, in general, it can be extended to a consistent admissible covering of $\{C_1, ..., C_n\}$). For instance, consider the sets $C_1 = \{p, q\}, C_2 = \{q, r\}, C_3 = \{p, s\}$, and let's assume that $S_1 = \{p\}$ (p is chosen at the first iteration) and $S_2 = \{p, q\}$. At this stage, the algorithm fails because S_2 is not admissible wrt $\{C_1, C_2\}$. Backtracking on the choice point of stage 2 yields $S'_2 = \{p, r\}$; at the third iteration nothing is added to S'_2 and it is output as a solution. However, if S_2 had not been rejected, at the next iteration it would be recognized as admissible wrt $\{C_1, C_2, C_3\}$. It seems that a correct solution has been lost. But the algorithm can still backtrack on the choice at the first stage: q can be chosen from C_1, i.e. $S'_1 = \{q\}$; nothing is added at the second iteration ($S'_1 \cap C_2 \neq \emptyset$), and finally p is added to S'_1 giving a second admissible solution $\{p, q\}$, thus recovering the apparently lost one.

This holds in general: it can be shown that

P2. If a set S_0 of literals is a consistent non-admissible covering of $\{C_1, ..., C_k\}$ and there exists $S \supseteq S_0$ that is admissible wrt $\{C_1, ..., C_k, C_{k+1}\}$, then there exists a consistent admissible covering $S_1 \subseteq S$ of $\{C_1, ..., C_k\}$.

Using P1 and P2, it can be proved that the algorithm is complete. In fact, if $\Gamma(T^*) = \{C_1, ..., C_n\}$, an inductive reasoning shows that, for any $i = 1, ..., n$, if S_i is a consistent and minimal covering of $C_1, ..., C_i$, then there exist choices of literals at iterations $1, ..., i$ such that S_i is built at stage i. The completeness proof, as well as proofs of P1 and P2, can be found in the technical report available at [7].

It is worth discussing alternative algorithms that would still be correct and complete. The simplest variant one can think of consists in first computing a consistent covering of $\{C_1, ..., C_n\}$ and then make it minimal eliminating unnecessary literals. Or, similarly, the algorithm presented above can be modified at line 9, in such a way that, if S_i is not admissible, then all redundant literals are removed from S_i before going on (without failing). A second alternative consists in failing at line 9 only if S_i is not admissible wrt $\{C_1, ..., C_i, ..., C_n\}$. Showing completeness of such variants is certainly easier, but in both cases it is very likely, even in simple cases, to produce several times the same solution. In order to avoid repetitions, each solution should be compared to all the previously found ones before it is given in output. The algorithm used by Logitest has not been proved to avoid such redundancies; however, in the experiments carried out, it did not happen to output the same solution more than once (run on an abductive problem with almost 5,000 solutions, no duplicates were found).

4 Concluding Remarks

The system described in this paper has been satisfactorily used by the author to design logical test items. Obviously, the user is assumed to have some basic logical notions in order to give adequate specifications in the input file.

Logitest can easily be improved by some routine work, in order to allow, for instance, a compact specification of some frequently used predicate properties (reflexivity, functionality, being an order relation, etc.), the possibility to define different belief sets, and the generation of literals that are consistent/inconsistent with the beliefs. A further step to make it more usable is the implementation of a graphical user interface.

Future work may finally include the study of common "logical misconceptions", so that belief sets, and consequently distractors, can be automatically generated.

Acknowledgements. The author thanks Serenella Cerrito for being willing to discuss the details of the abduction algorithm.

References

1. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers 35(8), 677–691 (1986)
2. Carneson, J., Delpierre, G., Masters, K.: Designing and managing multiple choice questions, http://web.uct.ac.za/projects/cbe/mcqman/mcqman01.html
3. Cialdea Mayer, M., Pirri, F.: First order abduction via tableau and sequent calculi. Bulletin of the Interest Group in Pure and Applied Logics (IGPL) 1, 99–117 (1993)
4. Clegg, V.L., Cashin, W.E.: Improving multiple-choice tests. Idea paper no. 16. Kansas State University: Center for Faculty Evaluation and Development (1986)
5. Eiter, T., Gottlob, G., Technische Universitt Wien: The complexity of logic-based abduction. Journal of the ACM 42, 3–42 (1995)
6. Leroy, X.: The Objective Caml system, release 3.11. Documentation and user's manual (2008), http://caml.inria.fr/
7. Logitest web page, http://cialdea.dia.uniroma3.it/logitest/

Goal-Directed Invariant Synthesis for Model Checking Modulo Theories

Silvio Ghilardi[1] and Silvio Ranise[2]

[1] Dipartimento di Informatica, Università degli Studi di Milano, Italy
[2] Dipartimento di Informatica, Università di Verona, Italy

Abstract. We are interested in automatically proving safety properties of infinite state systems. We present a technique for invariant synthesis which can be incorporated in backward reachability analysis. The main theoretical result ensures that (under suitable hypotheses) our method is guaranteed to find an invariant if one exists. We also discuss heuristics that allow us to derive an implementation of the technique showing remarkable speed-ups on a significant set of safety problems in parametrised systems.

1 Introduction

Backward reachability analysis has been widely adopted in model checking safety properties of infinite state systems (see, e.g., [1]). This verification procedure repeatedly computes pre-images of a set of unsafe states, usually obtained by complementing a safety property that a system should satisfy. Potentially infinite sets of states are represented by constraints so that pre-image computation can be done symbolically. A key advantage of backward reachability is to be *goal-directed*; the goal being the set of unsafe states from which pre-images are computed. Furthermore, safety properties for some classes of systems (e.g., broadcast protocols [8,6]) can be decided by backward reachability.

Despite these advantages, backward reachability can unnecessarily explore (large) portions of the symbolic state space of a system which are actually not required to verify the safety property under consideration. Even worse, in some cases the analysis may not detect a fix-point, thereby causing non-termination. In order to avoid visiting irrelevant parts of the symbolic state space during backward reachability, techniques for analyzing pre-images and guessing invariants have been devised (see, e.g., [5,15,9,4,13] to name a few). The success of these techniques depend crucially on the heuristics used to guess the invariants. Our framework is similar in spirit to [5], but employs techniques which are specific for our different intended application domains.

Along this line of research, we present a technique for interleaving pre-image computation and invariant synthesis which tries to eagerly prune irrelevant parts of the search space. Formally, we work in the framework of the model checking (based on Satisfiability) Modulo Theories approach of [10,12], where *array-based systems* have been introduced as an abstraction of several classes of infinite state

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 173–188, 2009.

systems (such as parametrized systems, lossy channels, and algorithms manipulating arrays). The **main result** (cf. Theorems 4.9 and 4.11) of the paper ensures that the technique *will find an invariant—provided one exists—under suitable hypotheses*, which are satisfied for important classes of array-based systems (e.g., mutual exclusion algorithms or cache coherence protocols). The key ingredient in the proof of the result is the model-theoretic notion of configuration and configuration ordering (introduced in [1] at an abstract level) which allows us to finitely characterize the search space of candidate invariants. Although the technique is developed for array-based systems, we believe that the underlying idea can be adapted to other symbolic approaches to model checking (e.g., [2,3]).

Plan of the paper. We briefly introduce the notion of array-based system (Sec. 2). We revisit the backward reachability procedure *as a Tableaux-like calculus* (Sec. 3) so as to give a firm basis for implementation. We show how invariants can help backward reachability (Sec. 4), recall the duality between this and the synthesis of invariants (Sec. 4.1), and describe how to interleave backward analysis and invariant synthesis (Sec. 4.2) together with some heuristics (Sec. 4.3). Finally (Sec. 5), we discuss how a prototype implementation of our techniques shows remarkable speed-ups. Full proofs and more examples can be found in the technical report [11].

2 Formal Preliminaries

We assume the usual syntactic (e.g., signature, variable, term, atom, literal, and formula) and semantic (e.g., structure, sub-structure, truth, satisfiability, and validity) notions of first-order logic (see, e.g., [7]). The equality symbol $=$ is included in all signatures considered below. A signature is *relational* if it does not contain function symbols and it is *quasi-relational* if its function symbols are all constants. An *expression* is a term, an atom, a literal, or a formula. Let $\underline{x}$ be a finite tuple of variables and Σ a signature; a $\Sigma(\underline{x})$-expression is an expression built out of the symbols in Σ where at most the variables in $\underline{x}$ may occur free (we will write $E(\underline{x})$ to emphasize that E is a $\Sigma(\underline{x})$-expression). Let $\underline{e}$ be a finite sequence of expressions and σ a substitution; $\underline{e}\sigma$ is the result of applying the substitution σ to each element of the sequence $\underline{e}$.

According to the current practice in the SMT literature [16], a *theory* T is a pair $(\Sigma, \mathcal{C})$, where Σ is a signature and $\mathcal{C}$ is a class of Σ-structures; the structures in $\mathcal{C}$ are the *models* of T. Below, we let $T = (\Sigma, \mathcal{C})$. A Σ-formula ϕ is *T-satisfiable* if there exists a Σ-structure $\mathcal{M}$ in $\mathcal{C}$ such that ϕ is true in $\mathcal{M}$ under a suitable assignment to the free variables of ϕ (in symbols, $\mathcal{M} \models \phi$); it is *T-valid* (in symbols, $T \models \varphi$) if its negation is T-unsatisfiable. Two formulae φ_1 and φ_2 are *T-equivalent* if $\varphi_1 \leftrightarrow \varphi_2$ is *T-valid.* The *satisfiability modulo the theory T ($SMT(T)$) problem* amounts to establishing the T-satisfiability of quantifier-free Σ-formulae.

T admits *quantifier elimination* iff for every formula $\varphi(\underline{x})$ one can compute a quantifier-free formula $\varphi'(\underline{x})$ such that $T \models \forall \underline{x}(\varphi(\underline{x}) \leftrightarrow \varphi'(\underline{x}))$. A theory $T = (\Sigma, \mathcal{C})$ is said to be *locally finite* iff Σ is finite and, for every finite set of variables $\underline{x}$, there are finitely many $\Sigma(\underline{x})$-terms $t_1, \ldots, t_{k_{\underline{x}}}$ such that for every further $\Sigma(\underline{x})$-term u, we have that $T \models u = t_i$ (for some $i \in \{1, \ldots, k_{\underline{x}}\}$). The terms $t_1, \ldots, t_{k_{\underline{x}}}$ are called *$\Sigma(\underline{x})$-representative terms*; if they are effectively computable from $\underline{x}$ (and t_i is computable from u), then T is said to be *effectively locally finite* (in the following, when we say 'locally finite', we in fact always mean 'effectively locally finite'). If Σ is relational or quasi-relational, then any Σ-theory T is locally finite. An *enumerated data-type theory T* is a theory in a quasi-relational signature whose class of models contains only a single finite Σ-structure $\mathcal{M} = (M, \mathcal{I})$ such that for every $m \in M$ there exists a constant $c \in \Sigma$ such that $c^{\mathcal{I}} = m$.

A *T-partition* is a finite set $C_1(\underline{x}), \ldots, C_n(\underline{x})$ of quantifier-free formulae such that $T \models \forall \underline{x} \bigvee_{i=1}^{n} C_i(\underline{x})$ and $T \models \bigwedge_{i \neq j} \forall \underline{x} \neg (C_i(\underline{x}) \wedge C_j(\underline{x}))$. A *case-definable extension* $T' = (\Sigma', \mathcal{C}')$ of a theory $T = (\Sigma, \mathcal{C})$ is obtained from T by applying (finitely many times) the following procedure: (i) take a T-partition $C_1(\underline{x}), \ldots, C_n(\underline{x})$ together with Σ-terms $t_1(\underline{x}), \ldots, t_n(\underline{x})$; (ii) let Σ' be $\Sigma \cup \{F\}$, where F is a "fresh" function symbol (i.e. $F \not\in \Sigma$) whose arity is equal to the length of $\underline{x}$; (iii) take as $\mathcal{C}'$ the class of Σ'-structures $\mathcal{M}$ whose Σ-reduct is a model of T and such that $\mathcal{M} \models \bigwedge_{i=1}^{n} \forall \underline{x}\ (C_i(\underline{x}) \to F(\underline{x}) = t_i(\underline{x}))$. Thus a case-definable extension T' of a theory T contains finitely many additional function symbols, called *case-defined functions*. It is not hard to translate any $SMT(T')$ problem into an equivalent $SMT(T)$-problem, by repeatedly applying the following transformation: given the quantifier free formula ϕ to be tested for T'-satisfiability, replace it by $\bigvee_i (C_i \sigma \wedge \phi_i)$, where ϕ_i is a formula obtained from ϕ by replacing each term of the kind $F\sigma$ by $t_i\sigma$ (the C_i's are the partition formulae for the case definition of F and the t_i's are the related 'value' terms).

From now on, we use many-sorted first-order logic. All notions introduced above can be easily adapted to a many-sorted framework. **In the rest of the paper, we fix** (i) a theory $T_I = (\Sigma_I, \mathcal{C}_I)$ for indexes whose only sort symbol is INDEX; (ii) a theory $T_E = (\Sigma_E, \mathcal{C}_E)$ for data whose only sort symbol is ELEM (the class $\mathcal{C}_E$ of models of this theory is usually a singleton). The **theory** $A_I^E = (\Sigma, \mathcal{C})$ **of arrays with indexes** I **and elements** E is obtained as the combination of T_I and T_E as follows: INDEX, ELEM, and ARRAY are the only sort symbols of A_I^E, the signature is $\Sigma := \Sigma_I \cup \Sigma_E \cup \{\_[\_]\}$ where $\_[\_]\ :\ \mathtt{ARRAY}, \mathtt{INDEX} \longrightarrow \mathtt{ELEM}$ (intuitively, $a[i]$ denotes the element stored in the array a at index i); a three-sorted structure $\mathcal{M} = (\mathtt{INDEX}^{\mathcal{M}}, \mathtt{ELEM}^{\mathcal{M}}, \mathtt{ARRAY}^{\mathcal{M}},\ \mathcal{I})$ is in $\mathcal{C}$ iff $\mathtt{ARRAY}^{\mathcal{M}}$ is the set of (total) functions from $\mathtt{INDEX}^{\mathcal{M}}$ to $\mathtt{ELEM}^{\mathcal{M}}$, the function symbol $\_[\_]$ is interpreted as function application, and $\mathcal{M}_I = (\mathtt{INDEX}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_I})$, $\mathcal{M}_E = (\mathtt{ELEM}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_E})$ are models of T_I and T_E, respectively (where $\mathcal{I}_{|\Sigma_X}$ is the restriction of the interpretation $\mathcal{I}$ to the symbols in Σ_X for $X \in \{I, E\}$).

Notational conventions. For the sake of brevity, we introduce the following notational conventions: d, e range over variables of sort ELEM, a over variables of sort ARRAY, i, j, k, and z over variables of sort INDEX. An underlined variable name abbreviates a tuple of variables of unspecified (but finite) length and, if $\underline{i} := i_1, \ldots, i_n$, the notation $a[\underline{i}]$ abbreviates the tuple of terms $a[i_1], \ldots, a[i_n]$. Possibly sub/super-scripted expressions of the form $\phi(\underline{i}, \underline{e}), \psi(\underline{i}, \underline{e})$ denote *quantifier-free $(\Sigma_I \cup \Sigma_E)$-formulae in which at most the variables $\underline{i} \cup \underline{e}$ occur.* Also, $\phi(\underline{i}, \underline{t}/\underline{e})$

(or simply $\phi(\underline{i}, \underline{t})$) abbreviates the substitution of the Σ-terms $\underline{t}$ for the variables $\underline{e}$. Thus, for instance, $\phi(\underline{i}, a[\underline{i}])$ denotes the formula obtained by replacing $\underline{e}$ with $a[\underline{i}]$ in the quantifier-free formula $\phi(\underline{i}, \underline{e})$.

3 Backward Reachability and Tableaux

Following [12], we focus on a particular yet large class of array-based systems corresponding to guarded assignments. A *(guarded assignment) array-based (transition) system (for (T_I, T_E))* is a triple $\mathcal{S} = (a, I, \tau)$ where (i) a is the *state* variable of sort `ARRAY`;[1] (ii) $I(a)$ is the *initial* $\Sigma(a)$-formula; and (iii) $\tau(a, a')$ is the *transition* $(\Sigma \cup \Sigma_D)(a, a')$-formula, where a' is a renamed copy of a and Σ_D is a finite set of case-defined function symbols not in $\Sigma_I \cup \Sigma_E$. Below, we also **assume $I(a)$ to be a $\forall^I$*-formula***, i.e. a formula of the form $\forall \underline{i}.\phi(\underline{i}, a[\underline{i}])$, and $\tau(a, a')$ **to be in *functional form***, i.e. a *disjunction of* formulae of the form

$$\exists \underline{i}\, (\phi_L(\underline{i}, a[\underline{i}]) \wedge \forall j\, a'[j] = F_G(\underline{i}, a[\underline{i}], j, a[j])) \tag{1}$$

where ϕ_L is the *guard* (also called the local component in [10]), and F_G is a case-defined function (called the *global* component). To understand why formulae (1) are in functional form, consider λ-abstraction; then, the sub-formula $\forall j\, a'[j] = F_G(\underline{i}, a[\underline{i}], j, a[j]))$ can be re-written as $a' = \lambda j.F_G(\underline{i}, a[\underline{i}], j, a[j])$. (By abuse of notation, any case-definable extension of $\mathcal{A}_I^E$ will be denoted by $\mathcal{A}_I^E$).

Given an array-based system $\mathcal{S} = (a, I, \tau)$ and a formula $U(a)$, (an instance of) the *safety problem* is to establish whether there exists a natural number n such that the formula

$$I(a_0) \wedge \tau(a_0, a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge U(a_n) \tag{2}$$

is $\mathcal{A}_I^E$-satisfiable. If there is no such n, then $\mathcal{S}$ is *safe* (w.r.t. U); otherwise, it is *unsafe* since the $\mathcal{A}_I^E$-satisfiability of (2) implies the existence of a run (of length n) leading the system from a state in I to a state in U. From now on, we **assume $U(a)$ to be a $\exists^I$*-formula***, i.e. a formula of the form $\exists \underline{i}.\phi(\underline{i}, a[\underline{i}])$.

A general approach to solve instances of the safety problem is based on computing the set of backward reachable states. For $n \geq 0$, the *n-pre-image* of a formula $K(a)$ is $Pre^0(\tau, K) := K$ and $Pre^{n+1}(\tau, K) := Pre(\tau, Pre^n(\tau, K))$, where

$$Pre(\tau, K) := \exists a'.(\tau(a, a') \wedge K(a')). \tag{3}$$

Given $\mathcal{S} = (a, I, \tau)$ and $U(a)$, the formula $Pre^n(\tau, U)$ describes the set of backward reachable states in n steps (for $n \geq 0$). At the n-th iteration of the loop, the *basic backward reachability algorithm*, depicted in Figure 1 (a), stores in the variable B the formula $BR^n(\tau, U) := \bigvee_{i=0}^n Pre^i(\tau, U)$ representing the set of states

[1] For simplicity (and without loss of generality), we limit ourselves to array-based systems having just one variable a of sort `ARRAY`. This limitation is however dropped in the examples, where in addition T_E may be many-sorted.

```
function BReach($U$ : $\exists^I$-formula)
1  $P \longleftarrow U$; $B \longleftarrow \bot$;
2  while ($P \wedge \neg B$ is $A_I^E$-sat.) do
3      if ($I \wedge P$ is $A_I^E$-sat.)
               then return unsafe;
4      $B \longleftarrow P \vee B$;
5      $P \longleftarrow Pre(\tau, P)$;
6  end
7  return (safe, $B$);
```

(a)

```
function SInv($U$ : $\exists^I$-formula)
1  $P \longleftarrow$ ChooseCover($U$); $B \longleftarrow \bot$;
2  while ($P \wedge \neg B$ is $A_I^E$-sat.) do
3      if ($I \wedge P$ is $A_I^E$-sat.)
               then return failure;
4      $B \longleftarrow P \vee B$;
5      $P \longleftarrow$ ChooseCover($Pre(\tau, P)$);
6  end
7  return (success, $\neg B$);
```

(b)

Fig. 1. The basic backward reachability (a) and the invariant synthesis (b) algorithms

which are backward reachable from the states in U in at most n steps (whereas the variable P stores the formula $Pre^n(\tau, U)$). While computing $BR^n(\tau, U)$, BReach also checks whether the system is unsafe (cf. line 3, which can be read as $I \wedge Pre^n(\tau, U)$ is A_I^E-satisfiable) or a fix-point has been reached (cf. line 2, which can be read as $\neg(BR^n(\tau, U) \rightarrow BR^{n-1}(\tau, U))$ is A_I^E-unsatisfiable or, equivalently, that $(BR^n(\tau, U) \rightarrow BR^{n-1}(\tau, U))$ is A_I^E-valid). When BReach returns the safety of the system (cf. line 7), the variable B stores the formula describing the set of states which are backward reachable from U which is also a fix-point. Indeed, for BReach (Figure 1 (a)) to be a true (possibly non-terminating) procedure, it is mandatory that (i) $\exists^I$-formulae are closed under pre-image computation and (ii) both the A_I^E-satisfiability test for safety (line 3) and that for fix-point (line 2) are effective.

Concerning (i), it is sufficient to recall the following result from [12].

Proposition 3.1. *Let* $K(a) \coloneqq \exists \underline{k}\, \phi(\underline{k}, a[\underline{k}])$ *and* $\tau(a, a') \coloneqq \bigvee_{h=1}^m \exists \underline{i}\; (\phi_L^h(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j. F_G^h(\underline{i}, a[\underline{i}], j, a[j]))$. *Then,* $Pre(\tau, K)$ *is* A_I^E*-equivalent to an (effectively computable)* $\exists^I$*-formula.*

The proof of Proposition 3.1 (see [12]) consists of applying simple logical manipulations to show that $Pre(\tau_h, K)$ is A_I^E-equivalent to the following $\exists^I$-formula, where τ_h is one of the m disjuncts of τ (cf. Proposition 3.1 above):

$$\exists \underline{i}\, \exists \underline{k}. (\phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F_G^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))) \tag{4}$$

where $\phi(\underline{k}, F_G^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))$ is the formula obtained from $\phi(\underline{k}, a'[\underline{k}])$ by replacing $a'[k_m]$ with $F_G^h(\underline{i}, a[\underline{i}], k_m, a[k_m])$ for $m = 1, ..., l$ and $\underline{k}$ is the tuple $k_1, \ldots, k_l$ (the F_G^h can then be eliminated as shown in Section 2). Notice that the existentially quantified prefix $\exists\ \underline{k}$ is augmented with $\exists\ \underline{i}$ in (4) with respect to K. Concerning (ii), observe that the formulae involved in the satisfiability checks are $I \wedge BR^n(\tau, K)$ and $BR^{n+1}(\tau, K) \wedge \neg BR^n(\tau, K)$. Since we have closure under pre-image computation, both formulae are of the form $\exists \underline{a}\; \exists \underline{i}\; \forall \underline{j}\; \psi(\underline{i}, \underline{j}, \underline{a}[\underline{i}], \underline{a}[\underline{j}])$ and are called $\exists^{A,I}\forall^I$*-sentences* [10].

Theorem 3.2 ([10]). *The A_I^E-satisfiability of $\exists^{A,I}\forall^I$-sentences is decidable if (i) T_I is locally finite and is closed under substructures; (ii) the $SMT(T_I)$ and $SMT(T_E)$ problems are decidable.*

Hypothesis (i) concerns the *topology* of the system (not the data manipulated by the components of the system) and it is satisfied in many practical cases, e.g., when the models of T_I are all finite sets, linear orders, graphs, forests, etc. For example, the topology of virtually any cache coherence protocol can be formalized by finite sets while that of mutual exclusion protocols by linear orders. Under assumption (i), it is possible to show (see [10]) that an $\exists^{A,I}\forall^I$-sentence is A_I^E-satisfiable iff it is satisfiable in a finite index model of A_I^E (a *finite index model* is a model $\mathcal{M}$ in which the set $\texttt{INDEX}^{\mathcal{M}}$ has finite cardinality). This suggests the following quantifier instantiation algorithm, which is indeed complete [10]. Let $\exists \underline{a}\ \exists \underline{i}\ \forall \underline{j}\ \psi(\underline{i}, \underline{j}, \underline{a}[\underline{i}], \underline{a}[\underline{j}])$ be an $\exists^{A,I}\forall^I$-sentence: first, consider the $\underline{i}$'s as Skolem constants and replace the $\underline{j}$'s with the representative $\underline{i}$-terms (by using the local finiteness of T_I); then, invoke the available SMT solver for checking the A_I^E-satisfiability of the resulting quantifier-free formula. The decidability of the $SMT(A_I^E)$ problem can be shown by using *generic combination techniques* from the decidability of those for $SMT(T_I)$ and $SMT(T_E)$ (see [10] for details).

We summarize our working hypotheses.

Assumption 3.3 *We fix an array-based system $\mathcal{S} = (a, I, \tau)$ such that the initial formula I is a $\forall^I$-formula, $\tau(a, a') := \bigvee_{h=1}^{m} \tau_h(a, a')$ where τ_h is a formula in functional form for $h = 1, ..., m$. We also assume that hypotheses (i)-(ii) of Theorem 3.2 are satisfied.*

3.1 Tableaux-Like Implementation of Backward Reachability

A naive implementation of the algorithm in Figure 1 (a) does not scale up. The main problem is the size of the formula $BR^n(\tau, U)$ which contains many redundant or unsatisfiable sub-formulae. We now discuss how Tableaux-like techniques can be used to circumvent these difficulties. We need one more definition: an $\exists^I$-formula $\exists i_1 \cdots \exists i_n \phi$ is said to be *primitive* iff ϕ is a conjunction of literals and is said to be *differentiated* iff ϕ contains as a conjunct the negative literal $i_k \neq i_l$ for all $1 \leq k < l \leq n$. By applying various distributive laws together with the rewriting rules

$$\exists j(i = j \wedge \theta) \rightsquigarrow \theta(i/j) \ \text{ and } \ \theta \rightsquigarrow (\theta \wedge i = j) \vee (\theta \wedge i \neq j) \tag{5}$$

it is possible to transform every $\exists^I$-formula into a disjunction of primitive differentiated ones.

We initialize our tableau with the $\exists^I$-formula $U(a)$ representing the set of unsafe states. The key observation is to revisit the computation of the preimage as the following inference rule (we use square brackets to indicate the applicability condition of the rule):

$$\frac{K \ \ [K \text{ is primitive differentiated}]}{Pre(\tau_1, K) \mid \cdots \mid Pre(\tau_m, K)}\ \mathsf{PreImg}$$

where $Pre(\tau_h, K)$ computes the $\exists^I$-formula which is logically equivalent to the pre-image of K w.r.t. τ_h (this is possible according to Proposition 3.1).

Since the $\exists^I$-formulae labeling the consequents of the rule PreImg may not be primitive and differentiated, we need the following Beta rule

$$\frac{K}{K_1 \mid \cdots \mid K_n}\ \mathsf{Beta}$$

where K is first transformed by eliminating the case-defined functions as explained in Section 2, and then by applying rewriting rules like (5) together with standard distributive laws, in order to get $K_1, \ldots, K_n$ which are primitive, differentiated and whose disjunction is A_I^E-equivalent to K.

By repeatedly applying the above rules, it is possible to build a tree whose nodes are labelled by $\exists^I$-formulae describing the set of backward reachable states. Indeed, it is not difficult to see that the disjunction of the $\exists^I$-formulae labelling all the nodes in the (potentially infinite) tree is A_I^E-equivalent to the (infinite) disjunction of the formulae $BR^n(\tau, U)$, where $\tau := \bigvee_{h=1}^m \tau_h$. Indeed, there is no need to fully expand our tree. For example, it is useless to apply the rule PreImg to a node ν labelled by an $\exists^I$-formula which is A_I^E-unsatisfiable as all the formulae labelling nodes in the sub-tree rooted at ν will also be A_I^E-unsatisfiable. This observation can be formalized by the following rule closing a branch in the tree (we mark the terminal node of a closed branch by ×):

$$\frac{K \quad [K \text{ is } A_I^E\text{-unsatisfiable}]}{\times}\ \mathsf{NotAppl}$$

This rule is effective since $\exists^I$-formulae are a subset of $\exists^{A,I}\forall^I$-sentences and the A_I^E-satisfiability of these formulae is decidable by Theorem 3.2.

According to procedure BReach, there are two more situations in which we can stop expanding a branch in the tree. One terminates the branch because of the safety test (cf. line 3 of Figure 1 (a)):

$$\frac{K \quad [I \wedge K \text{ is } A_I^E\text{-satisfiable}]}{\mathsf{UnSafe}}\ \mathsf{Safety}$$

Interestingly, if we label with τ_h the edge connecting a node labeled with K with that labeled with $Pre(\tau_h, K)$ when applying rule PreImg, then the transitions $\tau_{h_1}, ..., \tau_{h_e}$ labelling the edges in the branch terminated by UnSafe (from the leaf node to the root node) give a *bad trace*, i.e. a sequence of transitions leading the array-based system from a state satisfying I to one satisfying U. Again, rule UnSafe is effective since $I \wedge K$ is equivalent to an $\exists^{A,I}\forall^I$-sentence and its A_I^E-satisfiability is decidable by Theorem 3.2. The other situation in which one can close a branch corresponds to the fix-point test (cf. line 2 of Figure 1 (a))

$$\frac{K \quad [K \wedge \bigwedge\{\neg K' | K' \preceq K\} \text{ is } A_I^E\text{-unsatisfiable}]}{\times}\ \mathsf{FixPoint}$$

where $K' \preceq K$ means that K' is a primitive differentiated $\exists^I$-formula labeling a node preceding the node labeling K (nodes can be ordered according to the strategy for expanding the tree). Once more, this rule is effective since $K \wedge \bigwedge\{\neg K' | K' \preceq K\}$ can be straightforwardly transformed into an $\exists^{A,I}\forall^I$-sentence whose A_I^E-satisfiability is decidable by Theorem 3.2.

From the implementation viewpoint, further heuristics are needed, in order to reduce the instances needed for the satisfiability test of Theorem 3.2 and to trivialize the recognition of the unsatisfiable premise of the rule NotAppl.

4 Invariants and Backward Reachability

Termination of our tableaux calculus (and of the algorithm of Figure 1 (a)) is not guaranteed in general, but follows under certain restrictions covering important applications (see below). In the general case, nothing can be said because safety problems are undecidable.

Theorem 4.1. *The problem: "given an $\exists^I$-formula U, decide whether the array-based system $\mathcal{S}$ is safe w.r.t. U" is undecidable (even if T_E is locally finite).*

It is well-known that invariants are useful for pruning the search space of backward reachability procedures and may help either to obtain or to speed up termination.

Definition 4.2 (Safety invariants). *The $\forall^I$-formula $J(a)$ is a* safety invariant *for the safety problem consisting of the array-based system $\mathcal{S} = (a, I, \tau)$ and unsafe $\exists^I$-formula $U(a)$ iff the following conditions hold:*

(i) $A_I^E \models \forall a(I(a) \to J(a))$,
(ii) $A_I^E \models \forall a \forall a'(J(a) \wedge \tau(a, a') \to J(a'))$, *and*
(iii) $\exists a.(U(a) \wedge J(a))$ *is A_I^E-unsatisfiable.*

If we are not given the $\exists^I$-formula $U(a)$ and conditions (i)–(ii) hold, then $J(a)$ is an invariant for *$\mathcal{S}$.*

Checking whether conditions (i), (ii), and (iii) above hold can be reduced, by trivial logical manipulations, to the A_I^E-satisfiability of $\exists^{A,I}\forall^I$-formulae, which is decidable by Theorem 3.2. So, establishing whether a given $\forall^I$-formula $J(a)$ is a safety invariant can be completely automated.

Property 4.3. Let U be an $\exists^I$-formula. If there exists a safety invariant for U, then the array-based system $\mathcal{S} = (a, I, \tau)$ is safe with respect to U.

So, if we are given a suitable safety invariant, Property 4.3 can be used as the basis of the safety invariant method, which turns out to be more powerful than the basic Backward Reachability algorithm of Figure 1 (a):

Property 4.4. Let the procedure BReach in Figure 1(a) terminate on the safety problem consisting of the array-based system $\mathcal{S} = (a, I, \tau)$ and unsafe formula $U(a)$. If BReach returns (safe, B), then $\neg B$ is a safety invariant for U.

The converse of Proposition 4.4 do not hold: there might be a safety invariant even when BReach diverges, as illustrated by the following example.

Example 4.5. Let us consider a simple algorithm for inserting an element $b[0]$ into a sorted array $b[1], \ldots, b[n]$. Let Σ_I consist of one binary relation symbol S and one constant symbol 0 and T_I be the theory whose class of models consists of the substructures of the structure having the naturals as domain, with 0 interpreted in the obvious way, and S interpreted as the graph of the successor function. To simplify the matter, we shall use a two-sorted theory and two array variables. T_E is the two-sorted theory whose class of models consists of the single two-sorted structure given by the Booleans (with the constants $\top, \bot$ interpreted in the obvious way) and the rationals (with the standard ordering $<$). The array variable a is a Boolean flag, whereas the array variable b is the sorted numerical array where $b[0]$ is to be inserted. The initial $\forall^I$-formula is

$$\forall i\,(a[i] = \bot \leftrightarrow i \neq 0) \wedge \forall i_1, i_2\,(S(i_1, i_2) \to i_1 = 0 \vee b[i_1] \leq b[i_2])$$

saying that the elements in the array b, whose corresponding Boolean flag, is set to false are arranged in increasing order (namely, all except that at position 0). The transition has the following guard and global component:

$$\begin{array}{l}
\phi_L(i_1, i_2, a[i_1], a[i_2]) := S(i_1, i_2) \wedge a[i_1] = \top \wedge a[i_2] = \bot \wedge b[i_1] > b[i_2] \\
F_G(i_1, i_2, a[i_1], a[i_2], b[i_1], b[i_2], j) := \texttt{case of}\ \{ \quad \begin{array}{rcl} j = i_1 & : & \langle \top, b[i_2] \rangle, \\ j = i_2 & : & \langle \top, b[i_1] \rangle, \\ j \neq i_1 \wedge j \neq i_2 & : & \langle a[j], b[j] \rangle \ \}, \end{array}
\end{array}$$

which swaps two elements in the array b if their order is decreasing and sets the Boolean fields appropriately. The obvious correctness property is that there are no two values in decreasing order in the array b whose corresponding Boolean flags do not allow the transition to fire:

$$\exists i_1, i_2\,(S(i_1, i_2) \wedge \neg(a[i_1] = \top \wedge a[i_2] = \bot) \wedge b[i_1] > b[i_2]). \tag{6}$$

Unfortunately, BReach in Figure 1 (a) applied to (6) diverges. However, it is not difficult to see that a safety invariant for (6) exists and is given by the following formula:

$$\forall i, j.(S(i, j) \to \neg(a[i] = \bot \wedge a[j] = \top)) \tag{7}$$

saying that two adjacent indexes cannot have their Boolean flags set to $\bot$ and $\top$, respectively.

4.1 Synthesis of Invariants as the Dual of Backward Reachability

The main difficulty to exploit Property 4.3 is to find suitable $\forall^I$-formulae satisfying conditions (i)—(iii) of Definition 4.2. Unfortunately, the set of $\forall^I$-formulae which are candidates to become safety invariants is infinite. Such a search space can be dramatically restricted when T_E is locally finite, although it is still infinite because there is no bound on the length of the universally quantified prefix. To formalize this, we need to summarize some notions about pre-orders and configurations.

A *pre-order* $(P, \leq)$ is a set endowed with a reflexive and transitive relation; an *upset* of such a pre-order is a subset $U \subseteq P$ such that ($p \in U$ and $p \leq q$ imply $q \in U$). An upset U is *finitely generated* iff it is a finite union of cones, where a *cone* is an upset of the form $\uparrow p = \{q \in P \mid p \leq q\}$ for some $p \in P$. Two elements $p, q \in P$ are *incomparable* (*equivalent*) if neither (both) $p \leq q$ nor (and) $q \leq p$. A pre-order $(P, \leq)$ is a *well-quasi-ordering* (wqo) iff every upset of P is finitely generated (this is equivalent to the standard definition, see [10] for a proof).

An A_I^E*-configuration* (or, briefly, a configuration) is a pair $(s, \mathcal{M})$ such that s is an array of a finite index model $\mathcal{M}$ of A_I^E ($\mathcal{M}$ is omitted whenever it is clear from the context). We associate a Σ_I-structure s_I and a Σ_E-structure s_E with an A_I^E-configuration $(s, \mathcal{M})$ as follows: the Σ_I-structure s_I is simply the finite structure $\mathcal{M}_I$, whereas s_E is the smallest Σ_E-substructure of $\mathcal{M}_E$ containing the image of s (in other words, if $\mathtt{INDEX}^{\mathcal{M}} = \{c_1, \ldots, c_k\}$, then s_E is the smallest Σ_E-substructure containing $\{s(c_1), \ldots, s(c_k)\}$). Let s, s' be configurations: we say that $s' \leq s$ holds iff there are a Σ_I-embedding $\mu : s'_I \longrightarrow s_I$ and a Σ_E-embedding $\nu : s'_E \longrightarrow s_E$ such that the set-theoretical compositions of μ with s and of s' with ν are equal. In [10], termination of BReach is proved under the hypotheses that T_E is locally finite and the configuration order is a wqo. This implies the decidability of the safety problem for, among others, broadcast protocols and lossy channel systems and can be seen as the declarative counterpart of general results formulated within an algebraic framework (see, e.g., [1]). In the following, we show that using the notions of configuration and configuration order, it is possible to design a method for invariant synthesis.

Finitely generated upsets of configurations and $\exists^I$-formulae can be used interchangeably under a suitable assumption. Let $K(a)$ be an $\exists^I$-formula; we let $[\![K]\!] := \{(s, \mathcal{M}) \mid \mathcal{M} \models K(s)\}$.

Proposition 4.6 (Extended version of [10]). *Let T_E be locally finite. Finitely generated upsets of A_I^E-configurations coincide with sets of A_I^E-configurations of the kind $[\![K]\!]$, for some $\exists^I$-formula K. In particular, for each A_I^E-configuration s, there exists an $\exists^I$-formula K_s such that $[\![K_s]\!] = \uparrow s$.*

The notion of a basis for a configuration upset will be useful in the following.

Definition 4.7. *A* basis *for a finitely generated upset S (resp., for an $\exists^I$-formula K) is a minimal finite set $\{s_1, \ldots, s_n\}$ such that S (resp., $[\![K]\!]$) is equal to $\uparrow s_1 \cup \cdots \cup \uparrow s_n$.*

It is easy to see that two bases for the same upset are essentially the same, in the sense that they are formed by pairwise equivalent configurations. Our goal is to integrate the safety invariant method into the basic Backward Reachability algorithm of Figure 1(a). To this end, we introduce the notion of 'sub-reachability'.

Definition 4.8 (Subreachable configurations). *Suppose T_E is locally finite and let s be a configuration. A* predecessor *of s is any s' that belongs to a basis for $Pre(\tau, K_s)$. Let s, s' be configurations: s is* sub-reachable *from s' iff there exist configurations $s_0, \ldots, s_n$ such that (i) $s_0 = s$, (ii) $s_n = s'$, and (iii) either*

$s_{i-1} \leq s_i$ or s_{i-1} is a predecessor of s_i, for each $i = 1, \ldots, n$. If K is an $\exists^I$-formula, s is sub-reachable from *K iff s is sub-reachable from some s' taken from a basis of K.*

The following theorem is our main technical result.

Theorem 4.9. *Let T_E be locally finite. If there exists a safety invariant for U, then there are finitely many A_I^E-configurations $s_1, \ldots, s_k$ which are sub-reachable from U and such that $\neg(K_{s_1} \vee \cdots \vee K_{s_k})$ is also a safety invariant for U.*

The intuition underlying the theorem is as follows. Let us call 'finitely representable' an upset which is of the kind $[\![K]\!]$ for some $\exists^I$-formula K and let B be the set of backward reachable states. Usually B is infinite and it is finitely representable only in special cases (e.g., when the configuration ordering is a wqo like in the case of broad-cast protocols). Despite this, it may sometimes exist a set $B' \supseteq B$ which is finitely representable and whose complement is an invariant of the system. Theorem 4.9 ensures us to find such a B', if any. This is the case of Example 4.5 where not all configurations satisfying the negation of (7) are in B.

In practice, Theorem 4.9 suggests the following procedure to find the super-set B'. At each iteration of BReach, the algorithm represents symbolically in the variable B the configurations which are backward reachable in n steps; before computing the next pre-image of B, non deterministically replace some of the configurations in a basis of B with some sub-configurations and update B by a symbolic representation of the obtained upset. In this way, if an invariant exists, we are guaranteed to find it; otherwise, the process may diverge. This is so because the search space of the configurations which are sub-reachable in n steps is finite, although this number is infinite if no bound on n is fixed. To illustrate, the negation of (7) in Example 4.5 identifies sub-reachable only configurations. This shows that sub-reachability is crucial for Theorem 4.9 to hold.

The algorithm sketched above can be furtherly refined so as to obtain a completely symbolic method working with formulae without resorting to configurations. The key idea towards this result is to identify an $\exists^I$-formula which is the symbolic counterpart of the (sub-reachable) configurations $s_1, \ldots, s_k$ of the theorem above which can be directly computed from the available safety invariant for U. Formally, we introduce the following definition:

$$Min(\phi, a, \underline{i}) := \phi(\underline{i}, a[\underline{i}]) \wedge \bigwedge_{\sigma}(\phi(\underline{i}\sigma, a[\underline{i}\sigma]) \rightarrow \bigwedge_{i \in \underline{i}} \bigvee_{t} (t\sigma = i))$$

where $\phi(\underline{i}, a[\underline{i}])$ is a quantifier-free formula, t ranges over representative $\Sigma_I(\underline{i})$-terms, and σ ranges over the substitutions with domain $\underline{i}$ and co-domain included in the set of representative $\Sigma_I(\underline{i})$-terms. The formula $\exists \underline{i}.Min(\phi, a, \underline{i})$ is A_I^E-equisatisfiable to the $\exists^I$-formula $\exists \underline{i}.\phi(\underline{i}, a[\underline{i}])$; moreover if (as it often happens in applications) the signature Σ_I is relational and the formula $\phi(\underline{i}, a[\underline{i}])$ is differentiated, $Min(\phi, a, \underline{i})$ is A_I^E-equivalent to $\phi(\underline{i}, a[\underline{i}])$.

Proposition 4.10. *Let T_E be locally finite, $K := \exists \underline{i}.\phi(\underline{i}, a[\underline{i}])$ be an $\exists^I$-formula, and L be a further $\exists^I$-formula. The following two conditions are equivalent:*

(i) *for every s in a basis for K, there exists a configuration s' in a basis for L such that $s \leq s'$;*
(ii) *L is (up to A_I^E-equivalence) of the form $\exists \underline{i}, \underline{j}.\psi(\underline{i}, \underline{j}, a[\underline{i}], a[\underline{j}])$ for a quantifier-free formula ψ and*

$$\textit{if } A_I^E \models Min(\psi, a, \underline{i}\,\underline{j}) \rightarrow \theta(\underline{t}, a[\underline{t}]) \;\; \textit{then} \;\; A_I^E \models Min(\phi, a, \underline{i}) \rightarrow \theta(\underline{t}, a[\underline{t}]),$$

for all quantifier free $(\Sigma_E \cup \Sigma_I)$-formula θ and for all tuple of terms $\underline{t}(\underline{i})$ taken from the set of the representative $\Sigma_I(\underline{i})$-terms.

In the following, we will write $K \leq L$ whenever one of the (equivalent) conditions in Proposition 4.10 holds. Under the assumption that T_E is locally finite, it is possible to compute all the finitely many (up to A_I^E-equivalence) $\exists^I$-formulae K such that $K \leq L$. Furthermore, we say that K *covers* L iff both $K \leq L$ and $A_I^E \models L \rightarrow K$. Let ChooseCover($L$) be a procedure that returns (according to some criteria) one of the $\exists^I$-formulae K such that K covers L. We are now ready to give the procedure SInv in Figure 1 (b) for the computation of safety invariants and prove its correctness.

Theorem 4.11. *Let T_E be locally finite. Then, there exists a safety invariant for U iff the procedure* SInv *in Figure 1 (b) returns a safety invariant for U, for a suitable* ChooseCover *function.*

When ChooseCover(L) = L, i.e. ChooseCover is the identity (indeed, L covers L), the procedure SInv is the (exact) dual of BReach in Figure 1 (a) and, hence it can only return (the negation of) a symbolic representation of all backward reachable states as a safety invariant.

4.2 Integrating Invariant Synthesis within Backward Reachability

The main drawback of procedure SInv is the difficulty of defining an appropriate function ChooseCover. Although finite, the number of formulae covering a certain $\exists^I$-formula is so large that makes any implementation of SInv impractical. Instead, we prefer to study how to integrate the synthesis of invariants in the backward reachability algorithm in Figure 1 (a). The idea is to use invariants for the unsafe configuration U so as to prune the search space of the backward reachability algorithm. In our symbolic framework, at the n-th iteration of the loop of the procedure BReach, the set of backward reachable states is represented by the formula stored in the variable B (which is equivalent to $BR^n(\tau, U)$). So, 'pruning the search space of the backward reachability algorithm' amounts to disjoining the negation of the available invariants to B. In this way, the extra information encoded in the invariants makes the satisfiability test at line 2 (for fix-point checking) more likely to be successful and possibly decreasing the number of iterations of the loop.

Indeed, the problem is to synthesize such invariants. A way to do this is to consider the set B of reachable states, to extract an $\exists^I$-formula representing a set of sub-reachable configurations, and then checking whether this is an invariant. We assume the existence of a function ChooseSub that takes an $\exists^I$-formula P

and returns a (possibly empty) finite set S of $\exists^I$-formulae such that $K \leq P$ if $K \in S$. The formulae in S represent sub-reachable configurations that may contribute to an invariant in the sense of Theorem 4.9.

To summarize, it is possible to integrate the synthesis of invariants within the backward reachability algorithm by inserting between lines 4 and 5 in Figure 1 (a) the following instructions:

```
4'    foreach $CINV \in \mathsf{ChooseSub}(P)$ do
          if $\mathsf{BReach}(CINV) = (\mathsf{safe}, B_{CINV})$ then $B \longleftarrow B \vee \neg B_{CINV}$;
```

where $CINV$ stands for 'candidate invariant.' The resulting procedure will be indicated with BReach+Inv (notice that BReach is used here as a sub-procedure).

Proposition 4.11. *Let T_E be locally finite. If the procedure* BReach+Inv *terminates and returns safe (unsafe), then $\mathcal{S}$ is safe (unsafe) with respect to U.*

The procedure BReach+Inv is incomplete (in the sense that it is not guaranteed to terminate even in case a safety invariant exists), deterministic (no backtracking is required), and highly parallelizable (it is possible to run in parallel as many instances of BReach as formulae in the set returned by ChooseSub), and it performs well, as witnessed by the experimental evidence supplied in the next Section. In this way, invariant synthesis has become a powerful *heuristics* within a sophisticated version of the basic backward reachability algorithm. Furthermore, its integration in the Tableaux calculus of Sec. 3.1 is particularly easy: just use the calculus itself with some bounds on the resources, such as a limit on the depth of the tree to check if a candidate invariant is a true invariant.

4.3 Heuristics

There is a delicate trade-off between the number of candidate invariants produced by the function ChooseSub and their effects in pruning the search space of the basic backward reachability algorithm. More candidate invariants implies a higher probability of finding an invariant and, ultimately, to prune the search space. However, looking at line $4'$, it is evident that more candidate invariants implies many more calls to the basic backward reachability algorithms to establish if they are invariant or not. Indeed, on "simpler" candidate invariants, the procedure BReach is likely to perform well, i.e. to terminate in few iterations. The following two remarks are helpful in finding the right trade-off.

First, it is possible to limit the resources of the basic backward reachability algorithm BReach when invoking it at line $4'$; e.g., it is possible to bound the number of iterations of the loop or its run time. This allows us to avoid slowing down too much each iteration of the main loop in BReach+Inv.

The second remark concerns the implementation of the function ChooseSub when the theories T_I and T_E satisfy some additional requirements, which are often satisfied when modelling classes of parametrised systems such as mutual exclusion algorithms or cache coherence protocols. The goal of this discussion is to design a function ChooseSub returning few "simple" candidate invariants which are likely to become true invariants.

Claim. Let Σ_I be relational and let T_E be locally finite and admit elimination of quantifiers. (When T_I is the theory of all finite sets—this is appropriate for cache coherence protocols—or the theory of linear orders—this is appropriate for mutual exclusion algorithms—and T_E is the theory of an enumerated datatype, these assumptions are satisfied.) Let

$$L := \exists \underline{i}\,\underline{j}.(\psi_E(a[\underline{i}], a[\underline{j}]) \wedge \psi_I(\underline{i}, \underline{j}) \wedge \delta_I(\underline{i})) \tag{8}$$

be a primitive differentiated A_I^E-satisfiable $\exists^I$-formula such that (i) $\underline{i} \cap \underline{j} = \emptyset$, (ii) $\psi_E(\underline{e}, \underline{d})$ is a conjunction of Σ_E-literals; (iii) $\psi_I(\underline{i}, \underline{j})$ is a conjunction of Σ_I-literals; (iv) $\delta_I(\underline{i})$ is a maximal conjunction of $\Sigma_I(\underline{i})$-literals (i.e. for every $\Sigma(\underline{i})$-atom $A(\underline{i})$, δ_I contains either $A(\underline{i})$ or its negation). If

$$K := \exists \underline{i}\, (\delta_I(\underline{i}) \wedge \phi_E(a[\underline{i}])), \tag{9}$$

where $\phi_E(\underline{e})$ is T_E-equivalent to $\exists \underline{d}\, \psi_E(\underline{e}, \underline{d})$ (which is guaranteed to exist as T_E admits elimination of quantifiers), then K covers L and in particular $K \leq L$.

When ChooseSub is applied to a disjunction of primitive differentiated $\exists^I$-formulae, we need to transform each disjunct $P := \exists \underline{k}.\theta(\underline{k}, a[\underline{k}])$ to the form of (8) so as to obtain a candidate invariant. To do this, we can decompose $\underline{k}$ into two disjoint subsequences $\underline{i}$ and $\underline{j}$ such that $\underline{k} = \underline{i} \cup \underline{j}$ according to some criteria: if the conjunction of $\Sigma_I(\underline{i})$ literals occurring in θ is maximal, we get a candidate invariant by returning the corresponding $\exists^I$-formula (9). This is quite feasible in many concrete cases. For instance, quantifier elimination reduces to a trivial substitution if T_E is an enumerated datatype theory and the Σ_E-literals in θ are all positive. Maximality of θ is guaranteed (by differentiatedness) if T_I is the theory of finite sets; maximality of θ is also guaranteed if T_I is the theory of linear orders and $\underline{i} = i_1$ or ($\underline{i} = i_1, i_2$ and θ contains the atom $i_1 < i_2$).

5 Experiments and Discussion

To test the practical viability of our approach, we have implemented MCMT, a prototype tool which uses Yices (`http://yices.csl.sri.com`) as the backhand SMT solver. MCMT is the successor of the system in [12] which is not capable of solving almost any of the problems considered here. The starting point of our implementation is the Tableaux-like calculus of Section 3.1. As Yices is guaranteed to behave as a decision procedure on quantifier-free formulae only, universally quantified variables in $\exists^{A,I}\forall^I$-sentences are instantiated according to the procedure sketched after Theorem 3.2: this is required for the application of rules NotAppl, Safety, FixPoint. Invariants have been integrated in the basic backward reachability algorithm along the lines of Section 4.2.

As benchmarks, we have derived safety problems in our format from two sets of benchmarks in [2]: one is of mutual exclusion protocols (with 7 problems, cf. Table 1) and the other is of cache coherence protocols (with 9problems, cf.

Table 1. Mutual exclusion algorithms

	depth	#nodes	#calls	time	depth	#nodes	#calls	#inv	time
Bakery	9	29	221	0.104	7	8	129	5	**0.052**
Burns	14	57	497	0.216	2	2	59	3	**0.016**
Java M-lock	9	23	353	**0.156**	9	22	2390	1	0.772
Dijkstra	13	40	392	0.148	2	1	41	2	**0.012**
Dijkstra (rv)	14	138	6905	5.756	2	1	57	2	**0.016**
Szymanski	17	143	3266	2.208	11	22	1185	8	**0.288**
Szymanski (a)	23	2358	902017	24m19s	16	90	8547	16	**5.188**

Table 2. Cache coherence protocols

	depth	#nodes	#calls	time	depth	#nodes	#calls	#inv	time
Berkeley	2	1	102	**0.020**	2	1	190	0	0.032
Mesi	3	2	175	**0.032**	3	2	231	0	0.036
Moesi	3	2	304	**0.048**	3	2	384	0	0.052
Dec Firefly	4	4	163	**0.052**	4	4	222	0	0.068
Xerox P.D.	7	13	607	**0.288**	7	13	1059	0	0.432
Illinois	4	8	998	**0.196**	4	8	1114	0	0.216
Futurebus	4	19	1318	**0.460**	4	19	3824	0	1.096
German	26	2985	322335	**8m39s**	26	2856	544429	10	10m37s
German (pfs)	42	26004	3062165	176m51s	42	22808	2656282	40	**173m42s**

Table 2).[2] We used the theory of finite linear orders as T_I for mutual exclusion algorithms and the theory of finite sets as T_I for cache coherence protocols. The theory T_E for the various systems is the combination of an enumerated datatype theory for the control locations with theories for the data manipulated by the processes. A difficulty in the translation was the presence of global (i.e. universally quantified) guards which are not directly supported by our formalism. It is possible to eliminate universal quantifiers in guards (see [12] for details) by adopting the well-known *stopping failure model* (see, e.g., [14]) which is quite close to the approximate model in [2,3]. This is without loss of generality since establishing a safety property for the stopping failures model of a system trivially implies that the same property is enjoyed by the original system. The elimination of global guards can be easily mechanized as it is purely syntactic.

Columns 2-5 of both Tables report the statistics of our implementation of the procedure BReach while columns 6-10 show the results for BReach+Inv. (All timings are in seconds and obtained on a Pentium Dual-Core 3.4 GHz with 2 Gb Sdram). Table 1 clearly shows the usefulness of invariant search as the size of the problem grows. Table 2 seems to suggest that invariant search is useless or even detrimental to performances on cache coherence protocols. However, we remark that all these problems, except the German, are quite small and a brute

[2] The files containing such specifications and an executable of the tool are available at `http://homes.dsi.unimi.it/~ghilardi/mcmt`.

force search of the tiny search space (see the column '#nodes') is likely to be more successful. Furthermore, the overhead of searching for invariants can be eliminated by implementing a parallel version of the tool. Interestingly, there is some gain in using invariant synthesis on the last problem in this set (a difficult version of the German protocol [15], which is well-known to be a significant benchmark for verification tools). Although a comparative analysis is somewhat difficult in lack of a standard for the specifications of safety problems, we report that MCMT performs comparably with the model checker PFS [2] on small to medium sized problems and outperforms the latter on larger instances.

References

1. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite-state systems. In: Proc. of LICS, pp. 313–321 (1996)
2. Abdulla, P.A., Delzanno, G., Ben Henda, N., Rezine, A.: Regular model checking without transducers (On efficient verification of parameterized systems). In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 721–736. Springer, Heidelberg (2007)
3. Abdulla, P.A., Delzanno, G., Rezine, A.: Parameterized verification of infinite-state processes with global conditions. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 145–157. Springer, Heidelberg (2007)
4. Beyer, D., Henzinger, T.A., Majumdar, R., Rybalchenko, A.: Invariant Synthesis for Combined Theories. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 378–394. Springer, Heidelberg (2007)
5. Bradley, A.R., Manna, Z.: Property-Directed Incremental Invariant Generation. Formal Aspects of Computing (to appear, 2009)
6. Delzanno, G., Esparza, J., Podelski, A.: Constraint-based analysis of broadcast protocols. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 50–66. Springer, Heidelberg (1999)
7. Enderton, H.B.: A Mathematical Introduction to Logic. Academic Press, New York (1972)
8. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: Proc. of LICS, pp. 352–359. IEEE Computer Society Press, Los Alamitos (1999)
9. Flanagan, C., Qadeer, S.: Predicate abstraction for software verification. In: Proc. of POPL 2002, pp. 191–202. ACM Press, New York (2002)
10. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Towards SMT Model Checking of Array-Based Systems. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 67–82. Springer, Heidelberg (2008)
11. Ghilardi, S., Ranise, S.: Goal-directed Invariant Synthesis for Model Checking Modulo Theories. Technical Report RI325-09, Univ. di Milano (2009)
12. Ghilardi, S., Ranise, S., Valsecchi, T.: Light-Weight SMT-based Model-Checking. In: Proc. of AVOCS 2007-2008. ENTCS (2008)
13. Lahiri, S.K., Bryant, R.E.: Predicate Abstraction with Indexed Predicate. ACM Trans. on Comp. Logic 9(1) (2007)
14. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, San Francisco (1996)
15. Pnueli, A., Ruah, S., Zuck, L.D.: Automatic deductive verification with invisible invariants. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 82–97. Springer, Heidelberg (2001)
16. Ranise, S., Tinelli, C.: The SMT-LIB Standard: Version 1.2. Technical report, Dep. of Comp. Science, Iowa (2006), http://www.SMT-LIB.org/papers

Taming Displayed Tense Logics Using Nested Sequents with Deep Inference

Rajeev Goré, Linda Postniece, and Alwen Tiu

Logic and Computation Group,
College of Engineering and Computer Science,
The Australian National University

Abstract. We consider two sequent calculi for tense logic in which the syntactic judgements are nested sequents, i.e., a tree of traditional one-sided sequents built from multisets of formulae. Our first calculus **SKt** is a variant of Kashima's calculus for Kt, which can also be seen as a display calculus, and uses "shallow" inference whereby inference rules are only applied to the top-level nodes in the nested structures. The rules of **SKt** include certain structural rules, called "display postulates", which are used to bring a node to the top level and thus in effect allow inference rules to be applied to an arbitrary node in a nested sequent. The cut elimination proof for **SKt** uses a proof substitution technique similar to that used in cut elimination for display logics. We then consider another, more natural, calculus **DKt** which contains *no structural rules* (and no display postulates), but which uses *deep-inference* to apply inference rules directly at any node in a nested sequent. This calculus corresponds to Kashima's $S2Kt$, but with all structural rules absorbed into logical rules. We show that **SKt** and **DKt** are equivalent, that is, any cut-free proof of **SKt** can be transformed into a cut-free proof of **DKt**, and vice versa. We consider two extensions of tense logic, $Kt.S4$ and $S5$, and show that this equivalence between shallow- and deep-sequent systems also holds. Since deep-sequent systems contain no structural rules, proof search in the calculi is easier than in the shallow calculi. We outline such a procedure for the deep-sequent system **DKt** and its $S4$ extension.

1 Introduction

Belnap's Display Logic [2] (we prefer the term display calculi) is an extremely general proof-theoretical framework with the property that any sequent containing a particular formula occurrence A can be transformed into another sequent in which the occurrence of A is either the whole of the antecedent or the whole of the succedent, using only a subset of the rules called the display postulates. The occurrence of A is then said to be displayed. The most pleasing property of display calculi however is that if the rules of the display calculus enjoy eight easily checked conditions, then the calculus is guaranteed to obey cut-admissibility. That is, one single cut-admissibility proof suffices for all display calculi. This modularity makes it an excellent framework for designing sequent calculi for

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 189–204, 2009.

logics, particularly when we wish to mix and match the intuitionistic, modal, or substructural aspects of different logics into a new logic [17,7,6].

The generality of display calculi is obtained by adding a structural proxy for every logical connective and using residuation principles to implement the display property. For example, a display calculus for classical propositional logic usually contains Gentzen's "comma", but also a unary involutive structural connective "star" which allows us to flip structures from right/left to left/right of turnstile.

The main disadvantage of display calculi is that the display postulates can and must create large structures during the process of displaying a particular formula occurrence, making display calculi bad for backward proof-search. Display calculi also typically contain an explicit rule of contraction which duplicates complex structures when applied backwards, making it even harder to use them for backward proof search. A disciplined proof-theoretic methodology for transforming a display calculus into a more manageable traditional "contraction-free" calculus whilst preserving cut-admissiblity is therefore an important goal.

Our first step towards taming display calculi is to limit the structural connectives used in the calculi and consequently, the number of display postulates. Specifically, we work within display structures which can be viewed as a tree of traditional Gentzen's sequents, called *nested sequents*, which have been used previously by Kashima [12] and, independently, by Brünnler [3,4] to present several modal and tense logics. As in display calculi, Kashima's nested-sequent calculi contain "display-like" rules, called the *turn rules* in [12] and *residuation rules* in the display logic literature, which can be seen as tree transformations to bring a node in the nested sequent to the root. These residuation rules, and their interaction with structure contraction, are largely responsible for the difficulty in finding a proof search procedure for display-like calculi. Our second step is therefore to eliminate these rules without losing completeness.

We use Kashima's calculi for tense logics as a starting point for our proof theoretic (as opposed to the model-theoretic approach of Kashima) investigation into the broader problem of taming display calculi for proof search. We have recently shown that it is possible to tame the display calculus for Bi-Intuitionistic logic [8] by using nested sequents with a limited display property. The resulting calculus, $LBiInt_1$, still enjoys cut-admissibility. However, proof search for $LBiInt_1$ still suffers essentially the same problem as in display calculi, due to the presence of residuation and contraction on structures. In the same paper, we also show that these two problems can be eliminated entirely by a derived calculus $LBiInt_2$. However, the completeness proof of $LBiInt_2$ w.r.t. $LBiInt_1$ was done via a detour through a third calculus $GBiInt$ which is known to be semantically complete, and it was not clear how this methodology could be generalised to arbitrary display calculi for which the semantics may be unknown.

Here, we show that for some classical tense logics, residuation, seen as tree-transformations on nested sequents, and contraction (on general structures) are admissible if we allow a more liberal form of inference rule. Traditional rules of Gentzen's sequent calculus and display calculi apply only to formulae on the top level of a nested sequent. We shall call these rules "shallow inference".

Residuation and contraction become admissible once we allow *deep inference*, the ability to apply inference rules at any depth in a nested sequent.

The choice of classical tense logics as a case study is convenient because nested sequent calculi for these logics have already been given by Kashima [12]. But as we have noted earlier, Kashima's work is semantic based as there is no syntactic cut elimination procedure in his work. Thus our work is the first which shows direct syntactic cut elimination for a nested-sequent calculus for tense logic, and also the first which establishes a direct correspondence between proofs in a display-like calculus (with explicit residuation rules) and proofs in a contraction-free deep-inference calculus (with no explicit residuation rules).

We begin with Kashima's first system SKt which contains structural connectives (proxies) for $\Diamond$ and $\blacklozenge$ and contains explicit "turn" rules to capture the residuation conditions that hold between them. Kashima shows that SKt is sound with respect to the Kripke semantics for tense logic, but he does not prove cut-admissibility for this system. He instead gives another calculus S2Kt which allows rules to be applied at arbitrary depth, and shows that a sequent has a cut-free proof in SKt if it has a cut-free proof in S2Kt. In a second step, he shows that S2Kt minus cut is complete w.r.t. the Kripke semantics of tense logic, which together imply the completeness of SKt minus cut.

We first replace formula contraction with general contraction in Kashima's **SKt**, show that the resulting calculus enjoys a display property, and show that it also has cut-admissibility using an argument which is very similar to Belnap's cut-admissibility proof for display calculi. We then show that Kashima's S2Kt minus cut (in the form of our **DKt**) can be made contraction-free and that the display postulates of **SKt** are admissible in **DKt**, meaning that **DKt** can faithfully mimic cut-free **SKt**. We also show that **SKt** can mimic **DKt** by showing that all of the rules of **DKt** are actually derivable in **SKt** using the display property of **SKt**. We then show how to extend all these basic calculi to handle tense S4 and S5, but we are still not able to give a systematic method for converting the **SKt**-based calculi into the **DKt**-based calculi. Finally, we give a simple proof search strategy for **DKt**, as well as show how to add histories a là Heuerding to **DS4** for terminating proof search in the tense logic Kt.S4.

Due to space limit, most proofs are omitted, but they can be found in an extended version of the paper.

2 Tense Logic

To simplify presentation, we shall consider formulae of tense logic Kt which are in negation normal form (nnf), given by the following grammar:

$$A := a \mid \neg a \mid A \vee A \mid A \wedge A \mid \Box A \mid \blacksquare A \mid \Diamond A \mid \blacklozenge A.$$

where a ranges over atomic formulae and $\neg a$ is the negation of a. We shall denote with $\overline{A}$ the nnf of the negation of A. Implication can then be defined via negation: $A \to B = \overline{A} \vee B$. The axioms of minimal tense logic Kt are all the axioms of propositional logic, plus the following in their nnf form:

$$\begin{array}{llll}
w \Vdash \neg A & \text{iff } w \nVdash A & & \\
w \Vdash A \vee B & \text{iff } w \Vdash A \text{ or } w \Vdash B & w \Vdash A \wedge B & \text{iff } w \Vdash A \text{ and } w \Vdash B \\
w \Vdash \Box A & \text{iff } \forall u.\text{ if } wRu \text{ then } u \Vdash A & w \Vdash \Diamond A & \text{iff } \exists u.wRu \text{ and } u \Vdash A \\
w \Vdash \blacksquare A & \text{iff } \forall u.\text{ if } uRw \text{ then } u \Vdash A & w \Vdash \blacklozenge A & \text{iff } \exists u.uRw \text{ and } u \Vdash A
\end{array}$$

Fig. 1. Forcing of formulae

1. $A \rightarrow \Box\blacklozenge A = \overline{A} \vee \Box\blacklozenge A$
2. $A \rightarrow \blacksquare\Diamond A = \overline{A} \vee \blacksquare\Diamond A$
3. $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B) = \Diamond(A \wedge \overline{B}) \vee \Diamond\overline{A} \vee \Box B$
4. $\blacksquare(A \rightarrow B) \rightarrow (\blacksquare A \rightarrow \blacksquare B) = \blacklozenge(A \wedge \overline{B}) \vee \blacklozenge\overline{A} \vee \blacksquare B.$

The theorems of Kt are those that are generated from the above axioms and their substitution instances using the following rules:

$$\frac{A \quad \overline{A} \vee B}{B}\ MP \qquad \frac{A}{\Box A}\ Nec\Box \qquad \frac{A}{\blacksquare A}\ Nec\blacksquare$$

A *Kt-frame* is a pair $\langle W, R\rangle$, with W a non-empty set (of worlds) and $R \subseteq W \times W$. A Kt-model is a triple $\langle W, R, V\rangle$, with $\langle W, R\rangle$ a Kt frame and $V : Atm \rightarrow 2^W$ a valuation mapping each atom to the set of worlds where it is true.

For a world $w \in W$ and an atom $a \in Atm$, if $w \in V(a)$ then we write $w \Vdash a$ and say a is forced at w; otherwise we write $w \nVdash a$ and say a is rejected at w. Forcing and rejection of compound formulae is defined by mutual recursion in Figure 1. A Kt-formula A is valid iff it is forced by all worlds in all models, i.e. iff $w \Vdash A$ for all $\langle W, R, V\rangle$ and for all $w \in W$.

3 System SKt: A "Shallow" Calculus

We consider a right-sided proof system for tense logic where the syntactic judgment is a tree of multisets of formulae, called a *nested sequent*. Nested sequents have been used previously in proof systems for modal and tense logics [12,3].

Definition 1. *A* nested sequent *is a multiset*

$$\{A_1, \ldots, A_k, \circ\{\Gamma_1\}, \ldots, \circ\{\Gamma_m\}, \bullet\{\Delta_1\}, ..., \bullet\{\Delta_n\}\}$$

where $k, m, n \geq 0$, and each Γ_i and each Δ_j are themselves nested sequents.

We shall use the following notational conventions when writing nested sequents. We shall remove outermost braces, e.g., we write A, B, C instead of $\{A, B, C\}$. Braces for sequents nested inside $\circ\{\}$ or $\bullet\{\}$ are also removed, e.g., instead of writing $\circ\{\{A, B, C\}\}$, we write $\circ\{A, B, C\}$. When we juxtapose two sequents, e.g., as in Γ, Δ, we mean it is a sequent resulting from the multiset-union of Γ and Δ. When Δ is a singleton multiset, e.g., $\{A\}$ or $\{\circ\{\Delta'\}\}$, we simply write: Γ, A or $\Gamma, \circ\{\Delta'\}$. Since we shall only be concerned with nested sequents, we shall refer to nested sequents simply as sequents in the rest of the paper.

$$\frac{}{\Gamma, a, \bar{a}}\ id \qquad \frac{\Gamma, A \quad \Delta, \overline{A}}{\Gamma, \Delta}\ cut \qquad \frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \wedge B}\ \wedge \qquad \frac{\Gamma, A, B}{\Gamma, A \vee B}\ \vee$$

$$\frac{\Gamma, \Delta, \Delta}{\Gamma, \Delta}\ ctr \qquad \frac{\Gamma}{\Gamma, \Delta}\ wk \qquad \frac{\Gamma, \circ\{\Delta\}}{\bullet\{\Gamma\}, \Delta}\ rf \qquad \frac{\Gamma, \bullet\{\Delta\}}{\circ\{\Gamma\}, \Delta}\ rp$$

$$\frac{\Gamma, \bullet\{A\}}{\Gamma, \blacksquare A}\ \blacksquare \qquad \frac{\Gamma, \circ\{A\}}{\Gamma, \Box A}\ \Box \qquad \frac{\Gamma, \bullet\{\Delta, A\}}{\Gamma, \bullet\{\Delta\}, \blacklozenge A}\ \blacklozenge \qquad \frac{\Gamma, \circ\{\Delta, A\}}{\Gamma, \circ\{\Delta\}, \Diamond A}\ \Diamond$$

Fig. 2. System **SKt**

The above definition of sequents can also be seen as a special case of *structures* in display calculi, e.g., with ',' (comma), $\bullet$ and $\circ$ as structural connectives.

A *context* is a sequent with holes in place of formulae. A context with a single hole is written as $\Sigma[]$. Multiple-hole contexts are written as $\Sigma[] \cdots []$, or abbreviated as $\Sigma^k[]$ where k is the number of holes. We write $\Sigma^k[\Delta]$ to denote the sequent that results from filling the holes in $\Sigma^k[]$ uniformly with Δ.

The shallow proof system for Kt, called **SKt**, is given in Figure 2. This is basically Kashima's system (also called **SKt**) [12], but with a more general contraction rule (ctr), which allows contraction of arbitrary sequents. The modal fragment of **SKt** was also developed independently by Brünnler [3]. The general contraction rule is used to simplify our cut elimination proof, and as we shall see in Section 4, it can be replaced by formula contraction. System **SKt** can also be seen as a single-sided version of display calculus. The rules rp and rf are called the *residuation rules*. They are an example of *display postulates* commonly found in display calculus, and are used to bring a node in a nested sequent to the top level. The following is an analog of the display property of display calculus.

Proposition 1. *Let $\Sigma[\Delta]$ be a sequent. Then there exists a sequent Γ such that $\Sigma[\Delta]$ is derivable from Δ, Γ and vice versa, using only the rules rp and rf.*

Soundness and completeness. To prove soundness, we first show that each sequent has a corresponding Kt-formula, and then show that the rules of **SKt**, reading them top down, preserves validity of the formula corresponding to the premise sequent. Completeness is shown by simulating Hilbert's system for tense logic in **SKt**. The translation from sequents to formulae are given below. In the translation, we assume two logical constants $\bot$ ('false') and $\top$ ('true'). This is just a notational convenience, as the constants can be defined in a standard way, e.g., as $a \wedge \bar{a}$ and $a \vee \bar{a}$ for some fixed atomic proposition a. As usual, the empty disjunction denotes $\bot$ and the empty conjunction denotes $\top$.

Definition 2. *The function τ translates an* **SKt***-sequent*

$$\{A_1, \ldots, A_k, \circ\{\Gamma_1\}, \ldots, \circ\{\Gamma_m\}, \bullet\{\Delta_1\}, ..., \bullet\{\Delta_n\}\}$$

into the Kt-formula (modulo associativity and commutativity of $\vee$ and $\wedge$):

$$A_1 \vee \cdots \vee A_k \vee \Box\tau(\Gamma_1) \vee \cdots \vee \Box\tau(\Gamma_m) \vee \blacksquare\tau(\Delta_1) \vee \cdots \vee \blacksquare\tau(\Delta_n).$$

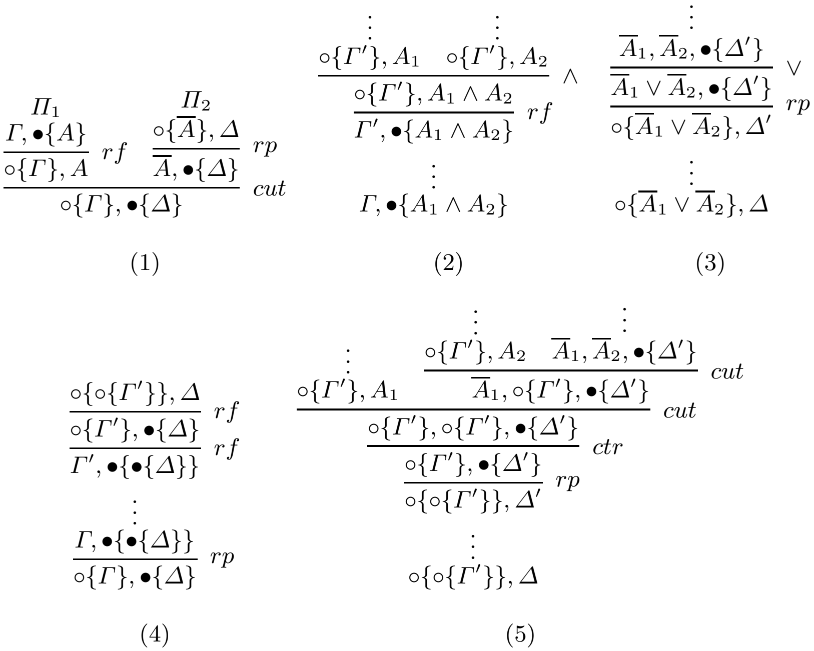

Π_1
$\Gamma, \bullet\{A\}$
$\circ\{\Gamma\}, A$
rf
Π_2
$\circ\{\overline{A}\}, \Delta$
$\overline{A}, \bullet\{\Delta\}$
rp
cut
$\circ\{\Gamma\}, \bullet\{\Delta\}$
(1)
$\circ\{\Gamma'\}, A_1$
$\circ\{\Gamma'\}, A_2$
$\wedge$
$\circ\{\Gamma'\}, A_1 \wedge A_2$
rf
$\Gamma', \bullet\{A_1 \wedge A_2\}$
$\Gamma, \bullet\{A_1 \wedge A_2\}$
(2)
$\overline{A}_1, \overline{A}_2, \bullet\{\Delta'\}$
$\vee$
$\overline{A}_1 \vee \overline{A}_2, \bullet\{\Delta'\}$
rp
$\circ\{\overline{A}_1 \vee \overline{A}_2\}, \Delta'$
$\circ\{\overline{A}_1 \vee \overline{A}_2\}, \Delta$
(3)
$\circ\{\circ\{\Gamma'\}\}, \Delta$
rf
$\circ\{\Gamma'\}, \bullet\{\Delta\}$
rf
$\Gamma', \bullet\{\bullet\{\Delta\}\}$
$\Gamma, \bullet\{\bullet\{\Delta\}\}$
rp
$\circ\{\Gamma\}, \bullet\{\Delta\}$
(4)
$\circ\{\Gamma'\}, A_2$
$\overline{A}_1, \overline{A}_2, \bullet\{\Delta'\}$
cut
$\circ\{\Gamma'\}, A_1$
$\overline{A}_1, \circ\{\Gamma'\}, \bullet\{\Delta'\}$
cut
$\circ\{\Gamma'\}, \circ\{\Gamma'\}, \bullet\{\Delta'\}$
ctr
$\circ\{\Gamma'\}, \bullet\{\Delta'\}$
rp
$\circ\{\circ\{\Gamma'\}\}, \Delta'$
$\circ\{\circ\{\Gamma'\}\}, \Delta$
(5)

Fig. 3. Some derivations in **SKt**

Theorem 1. *A Kt-formula A is valid iff A is* **SKt***-derivable.*

Cut elimination. The main difficulty in proving cut elimination for **SKt** is in finding the right cut reduction for some cases involving the rules rp and rf. For instance, consider the derivation (1) in Figure 3. It is not obvious that there is a cut reduction strategy that works locally without generalizing the cut rule to, e.g., one which allows cut on any sub-sequent in a sequent. Instead, we shall follow a global cut reduction strategy similar to that used in cut elimination for display logics. The idea is that, instead of permuting the cut rule locally, we trace the cut formula A (in Π_1) and $\overline{A}$ (in Π_2), until they both become principal in their respective proofs, and then apply the cut rule(s) at that point on smaller formulae. Schematically, our simple strategy can be illustrated as follows: Suppose that Π_1 and Π_2 are, respectively, derivation (2) and (3) in Figure 3, that $A = A_1 \wedge A_2$ and there is a single instance in each proof where the cut formula is used. To reduce the cut on A, we first transform Π_1 by uniformly substituting $\bullet\{\Delta\}$ for A in Π_1 (see derivation (4) in Figure 3). We then prove the open leaf $\{\circ\{\circ\{\Gamma'\}\}, \Delta\}$ by uniformly substituting $\circ\{\Gamma'\}$ for A in Π_2 (see derivation (5) in Figure 3). Notice that the cuts on A_1 and A_2 introduced in the proof above are on smaller formulae than A.

The above simplified explanation implicitly assumes that a uniform substitution of a formula (or formulae) in a proof results in a well-formed proof, and that the cut formulae are not contracted. The precise statement of the proof substitution idea becomes more involved once these aspects are taken into account. The formal statement is given in the lemma below. We use the notation $\vdash_S \Gamma$ to denote that the sequent Γ is provable in the proof system S. We write $\vdash_S \Pi : \Gamma$ when we want to be explicit about the particular proof Π of Γ. The *cut rank* of an instance of cut is defined as usual, as the size of the cut formula. The cut rank of a proof Π, denoted with $cr(\Pi)$, is the largest cut rank of the cut instances in Π (or zero, if there are no cuts in Π). Given a formula A, we denote with $|A|$ its size. Given a proof Π, we denote with $|\Pi|$ its height, i.e., the length of a longest branch in the proof tree of Π.

Lemma 1. *Let A be a non-atomic formula. Suppose $\vdash_{\mathbf{SKt}} \Pi_1 : \Delta, \overline{A}$ and $\vdash_{\mathbf{SKt}} \Pi_2 : \Sigma^k[A]$, for some $k \geq 1$, and the cut ranks of Π_1 and Π_2 are smaller than $|A|$. Then there exists a proof Π such that $\vdash_{\mathbf{SKt}} \Pi : \Sigma^k[\Delta]$ and $cr(\Pi) < |A|$.*

Theorem 2. *Cut elimination holds for* **SKt***.*

4 System DKt: A Contraction-Free Deep-Sequent Calculus

We now consider another sequent system which uses *deep inference*, where rules can be applied directly to any node within a nested sequent. We call this system **DKt**, and give its inference rules in Figure 4. Note that there are no structural rules in **DKt**, and the contraction rule is absorbed into the logical rules. Notice that, reading the logical rules bottom up, we keep the principal formulae in the premise. This is actually not neccessary for some rules (e.g., ■, $\wedge$, etc.), but this form of rule allows for a better accounting of formulae in our saturation-based proof search procedure (see Section 6).

The following intuitive observation about **DKt** rules will be useful later: Rules in **DKt** are characterized by propagations of formulae across different nodes in a nested sequent tree. The shape of the tree is not affected by these propagations, and the only change that can occur to the tree is the creation of new nodes (via the introduction rules ■ and □).

System **DKt** corresponds to Kashima's $S2Kt$ [12], but with the contraction rule absorbed into the logical rules. Kashima shows that **DKt** proofs can be encoded into **SKt**, essentially due to the display property of **SKt** (Proposition 1) which allows displaying and undisplaying of any node within a nested sequent. Kashima also shows that **DKt** is complete for tense logic, via semantic arguments. We prove a stronger result: every cut-free **SKt**-proof can be transformed into a **DKt**-proof, hence **DKt** is complete and cut is admissible in **DKt**.

To translate cut-free **SKt**-proofs into **DKt**-proofs, we show that all structural rules of **SKt** are height-preserving admissible in **DKt**, as stated next.

Lemma 2 (Admissibility of weakening). *Suppose $\vdash_{\mathbf{DKt}} \Pi : \Sigma[\Gamma]$. Then for every Δ, there exists Π' such that $\vdash_{\mathbf{DKt}} \Pi' : \Sigma[\Gamma, \Delta]$ and $|\Pi'| \leq |\Pi|$.*

$$
\begin{array}{ccc}
\dfrac{}{\varSigma[a,\bar{a}]}\ id & \dfrac{\varSigma[A \wedge B, A] \quad \varSigma[A \wedge B, B]}{\varSigma[A \wedge B]}\ \wedge & \dfrac{\varSigma[A \vee B, A, B]}{\varSigma[A \vee B]}\ \vee \\[2ex]
\dfrac{\varSigma[\blacksquare A, \bullet\{A\}]}{\varSigma[\blacksquare A]}\ \blacksquare & \dfrac{\varSigma[\bullet\{\varDelta, A\}, \blacklozenge A]}{\varSigma[\bullet\{\varDelta\}, \blacklozenge A]}\ \blacklozenge_1 & \dfrac{\varSigma[\circ\{\varDelta, A\}, \Diamond A]}{\varSigma[\circ\{\varDelta\}, \Diamond A]}\ \Diamond_1 \\[2ex]
\dfrac{\varSigma[\Box A, \circ\{A\}]}{\varSigma[\Box A]}\ \Box & \dfrac{\varSigma[\circ\{\varDelta, \blacklozenge A\}, A]}{\varSigma[\circ\{\varDelta, \blacklozenge A\}]}\ \blacklozenge_2 & \dfrac{\varSigma[\bullet\{\varDelta, \Diamond A\}, A]}{\varSigma[\bullet\{\varDelta, \Diamond A\}]}\ \Diamond_2
\end{array}
$$

Fig. 4. A contraction-free deep-sequent system

The proofs for the following lemmas that concern structural rules that change the shape of the tree of a nested sequent share similarities. That is, the only interesting cases in the proofs are those that concern propagation of formulae across different nodes in a nested sequent. We show here an interesting case in the proof for the admissibility of display postulates.

Lemma 3 (Admissibility of display postulates). *If* $\vdash_{\mathbf{DKt}} \varPi : \varGamma, \bullet\{\varDelta\}$ *then there exists* $\varPi'$ *such that* $\vdash_{\mathbf{DKt}} \varPi' : \circ\{\varGamma\}, \varDelta$ *and* $|\varPi'| \leq |\varPi|$.

Proof. By induction on $|\varPi|$. The non-trivial cases are when there is an exchange of formulae between $\varGamma$ and $\varDelta$. One example is when $\varPi$ is as shown below left. Then $\varPi'$ is as shown below right where $\varPi'_1$ is obtained from induction hypothesis:

$$
\dfrac{\begin{array}{c}\varPi_1 \\ \varGamma', \blacklozenge A, \bullet\{A, \varDelta\}\end{array}}{\varGamma', \blacklozenge A, \bullet\{\varDelta\}}\ \blacklozenge_1
\qquad\qquad
\dfrac{\begin{array}{c}\varPi'_1 \\ \circ\{\varGamma', \blacklozenge A\}, A, \varDelta\end{array}}{\circ\{\varGamma', \blacklozenge A\}, \varDelta}\ \blacklozenge_2
$$

Lemma 4 (Admissibility of display postulates). *If* $\vdash_{\mathbf{DKt}} \varPi : \varGamma, \circ\{\varDelta\}$ *then there exists* $\varPi'$ *such that* $\vdash_{\mathbf{DKt}} \varPi' : \bullet\{\varGamma\}, \varDelta$ *such that* $|\varPi'| \leq |\varPi|$.

Lemma 5 (Admissibility of contraction). *If* $\vdash_{\mathbf{DKt}} \varPi : \varSigma[\varDelta, \varDelta]$ *then there exists* $\varPi'$ *such that* $\vdash_{\mathbf{DKt}} \varPi' : \varSigma[\varDelta]$ *and* $|\varPi'| \leq |\varPi|$.

Theorem 3. *For every sequent* $\varGamma$, $\vdash_{\mathbf{SKt}} \varGamma$ *if and only if* $\vdash_{\mathbf{DKt}} \varGamma$.

A consequence of Theorem 3 is that the general contraction rule in **SKt** can be replaced by formula contraction. This can be proved as follows: take a cut-free proof in **SKt**, translate it to **DKt** and then translate it back to **SKt**. Since general contraction is admissible in **DKt**, and since the translation from **DKt** to **SKt** does not use general contraction (only formula contraction), we can effectively replace the general contraction in **SKt** with formula contraction.

An interesting feature of **DKt** is that in a proof of a sequent, the 'color' of a (formula or structural) connective does not change when moving from premise to conclusion or vice versa. Let us call a formula (a sequent, a rule) *purely modal* if it contains no black connectives. It is easy to see that if a purely modal formula (sequent) is provable in **DKt**, then it is provable using only purely modal rules. Let $\mathbf{DK} = \{id, \wedge, \vee, \Box, \Diamond_1\}$, i.e., it is the set of purely modal rules of **DKt**. The above observation leads to the following "separation" result:

Theorem 4. *For every modal formula φ, $\vdash_{\mathbf{DK}} \varphi$ iff φ is a theorem of K.*

This completeness result for **DK** is known from [3]; what we show here is how it can be derived as a consequence of completeness of **DKt**.

5 Proof Systems for Some Extensions of Tense Logic

We now consider extensions of tense logic with some modal axioms. We show that, for each extension, there is a shallow system that modularly extends **SKt** for which cut elimination holds. By modular extension we mean that the rules of the extended systems are the rules of **SKt** plus some structural rules that are derived directly from the modal axioms. We then show that for each extension, there is also a corresponding deep-inference system which is equivalent to the shallow one. Again, as with **DKt**, the rules for the deep system are characterized by propagations of formulae across different nodes in the nested sequents. However, the design of the rules for the deep system is not as modular as its shallow counterpart, since it needs to take into account the closure of the axioms.

Cut elimination holds for all the extensions discussed in the following. Their proofs are omitted as they are a straightforward adaptation of the cut elimination proof for **SKt**. This is because the proof substitution technique used for cut elimination in **SKt** relies on rule applications being invariant under formula substitution. More precisely, all the additional structural rules that we shall consider have the following property: If there is an instance of a structural rule ρ (below left) then instantiating the occurrences of A in the multi-context Σ_1 and Σ_2 with any structure Δ yields a valid instance of ρ (below right):

$$\frac{\Sigma_2^k[A]}{\Sigma_1^k[A]}\,\rho \qquad\qquad \frac{\Sigma_2^k[\Delta]}{\Sigma_1^k[\Delta]}\,\rho.$$

Hence the proof substitution technique for cut elimination goes through essentially unchanged for the extended logic. This property of the structural rules is similar to Belnap's condition (C6) for cut elimination for display logics [2].

A *primitive axiom* is an axiom of the form $A \rightarrow B$ where both A and B are built using propositional variables, $\wedge$, $\vee$, $\Diamond$, and $\blacklozenge$. Kracht [13] shows that any extension of tense logic with primitive axioms has a display calculus which enjoys cut elimination. He shows that any such axiom can be turned into a left structural rule. The axioms we consider next are contrapositives of primitive axioms, so Kracht's translation from axioms to structural rules in our formalism gives right structural rules. We illustrate here a few cases of primitive axioms for which one can also get corresponding deep sequent systems.

Modal tense logic S4. Consider an extension of **SKt** with the following axioms:

$$T : \Box A \rightarrow A \quad \blacksquare A \rightarrow A \qquad 4 : \Box A \rightarrow \Box\Box A \quad \blacksquare A \rightarrow \blacksquare\blacksquare A.$$

$$\frac{\Sigma[\blacklozenge A, A]}{\Sigma[\blacklozenge A]}\,T_a \qquad \frac{\Sigma[\blacklozenge A, \bullet\{\blacklozenge A, \Delta\}]}{\Sigma[\blacklozenge A, \bullet\{\Delta\}]}\,4_a \qquad \frac{\Sigma[\Diamond A, \circ\{\Diamond A, \Delta\}]}{\Sigma[\Diamond A, \circ\{\Delta\}]}\,4_c$$

$$\frac{\Sigma[\Diamond A, A]}{\Sigma[\Diamond A]}\,T_b \qquad \frac{\Sigma[\circ\{\Delta, \blacklozenge A\}, \blacklozenge A]}{\Sigma[\circ\{\Delta, \blacklozenge A\}]}\,4_b \qquad \frac{\Sigma[\bullet\{\Delta, \Diamond A\}, \Diamond A]}{\Sigma[\bullet\{\Delta, \Diamond A\}]}\,4_d$$

Fig. 5. Additional propagation rules for **DS4**

$$\frac{\Sigma[\blacklozenge A, \circ\{\blacklozenge A, \Delta\}]}{\Sigma[\blacklozenge A, \circ\{\Delta\}]}\,5_a \qquad \frac{\Sigma[\circ\{\Delta, \Diamond A\}, \Diamond A]}{\Sigma[\circ\{\Delta, \Diamond A\}]}\,5_b$$

$$\frac{\Sigma[\Diamond A, \bullet\{\Diamond A, \Delta\}]}{\Sigma[\Diamond A, \bullet\{\Delta\}]}\,5_c \qquad \frac{\Sigma[\bullet\{\Delta, \blacklozenge A\}, \blacklozenge A]}{\Sigma[\bullet\{\Delta, \blacklozenge A\}]}\,5_d$$

Fig. 6. Additional propagation rules for **DS5**

These axioms translate into the following structural rules, whose soundness is immediately derivable from the axioms:

$$\frac{\Gamma, \bullet\{\Delta\}}{\Gamma, \Delta}\,T_p \qquad \frac{\Gamma, \circ\{\Delta\}}{\Gamma, \Delta}\,T_f \qquad \frac{\Gamma, \bullet\{\Delta\}}{\Gamma, \bullet\{\bullet\{\Delta\}\}}\,4_p \qquad \frac{\Gamma, \circ\{\Delta\}}{\Gamma, \circ\{\circ\{\Delta\}\}}\,4_f$$

Definition 3 (System SS4). *System* **SS4** *is* **SKt** *plus* T_p, T_f, 4_p *and* 4_f.

Theorem 5. *Cut elimination holds for* **SS4**.

Definition 4 (System DS4). *System* **DS4** *is* **DKt** *plus the propagation rules given in Figure 5.*

Some of the modal rules of **DS4** coincide with Brünnler's rules for T and 4 in [3]. The rules of **DS4** can be shown to be derivable in **SS4**.

Lemma 6. *Every rule of* **DS4** *is derivable in* **SS4**.

To prove the equivalence of **SS4** and **DS4**, we need to prove the analogs of Lemma 2 – 5. These are again a straightforward adaptation of the previous proofs, and are omitted here. Additionally, we need to show that the structural rules for the axioms T and 4 are also admissible in **DS4**. The principle behind the proofs of admissibility for these structural rules is again the same; the non-trivial cases we need to consider are those that concern propagation of formulae across structures affected by the structural rules.

Theorem 6. *For every* Γ, *we have* $\vdash_{\mathbf{SS4}} \Gamma$ *if and only if* $\vdash_{\mathbf{DS4}} \Gamma$.

Modal tense logic S5. We can obtain S5 from **SS4** by collapsing $\Box$ and $\blacksquare$. That is, the symmetry axiom $B : A \rightarrow \Box\Diamond A$ splits into two axioms given below,

```
Function Prove (Sequent $\Xi$) : Bool

1. Let $T = tree(\Xi)$
2. If the $id$ rule is applicable to any node in $T$, return $True$
3. Else if there is some node $\Theta \in T$ that is not saturated
   (a) If $A \vee B \in \Theta$ and $A \notin \Theta$ or $B \notin \Theta$ then let $\Xi_1$ be the premise of the $\vee$ rule
       applied to $A \vee B \in \Theta$. Return $Prove(\Xi_1)$.
   (b) If $A \wedge B \in \Theta$ and $A \notin \Theta$ and $B \notin \Theta$ then let $\Xi_1$ and $\Xi_2$ be the premises
       of the $\wedge$ rule applied to $A \wedge B \in \Theta$. Return $True$ iff $Prove(\Xi_1) = True$ and
       $Prove(\Xi_2) = True$.
4. Else if there is some node $\Theta \in T$ that is not realised, i.e. some $B = \Box A$ ($B = \blacksquare A$)
   is not realised
   (a) Let $\Xi_1$ be the premise of the $\Box$ ($\blacksquare$) rule applied to $B \in \Theta$. Return $Prove(\Xi_1)$.
5. Else if there is some node $\Theta$ that is not propagated
   (a) Let $\rho$ be the rule corresponding to the requirement of Definition 9 that is not
       met, and let $\Xi_1$ be the premise of $\rho$. Return $Prove(\Xi_1)$.
6. Else return $False$
```

Fig. 7. Proof search strategy for **DKt**

which translate straightforwardly into two structural rules.

$$B1 : \blacksquare A \to \Box A \qquad \frac{\Gamma, \bullet\{\Delta\}}{\Gamma, \circ\{\Delta\}}\, B_1 \qquad\qquad B2 : \Box A \to \blacksquare A \qquad \frac{\Gamma, \circ\{\Delta\}}{\Gamma, \bullet\{\Delta\}}\, B_2$$

Definition 5 (System SS5). *System* **SS5** *is* **SS4** *plus the rules B_1 and B_2.*

Theorem 7. *Cut elimination holds for* **SS5**.

Definition 6 (System DS5). *System* **DS5** *is* **DS4** *plus the propagation rules given in Figure 6.*

Lemma 7. *Every rule of* **DS5** *is derivable in* **SS5**.

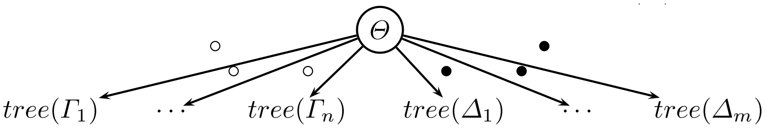
We can prove the analogs of Lemma 2 – 5 and admissibility of the rules corresponding to the axioms of **SS4** and structural rules B_1 and B_2. Note that **DS5** captures $S5 = KT45$ rather than $S5 = KT4B$.

Theorem 8. *For every Γ, we have $\vdash_{\mathbf{SS5}} \Gamma$ if and only if $\vdash_{\mathbf{DS5}} \Gamma$.*

6 Proof Search

We can devise terminating proof search strategies for our deep sequent calculi. While traditional tableaux methods operate on a single node at a time, our proof search strategies will consider the whole tree. Following Kashima, first we define a mapping from sequents to trees.

A *node* is a set of formulae. A *tree* is a node with 0 or more children, where each child is a tree, and each child is labelled as either a $\circ$-child, or a $\bullet$-child. Given a sequent $\Xi = \Theta, \circ\{\Gamma_1\}, \cdots, \circ\{\Gamma_n\}, \bullet\{\Delta_1\}, \cdots, \bullet\{\Delta_m\}$, where Θ is a set of formulae and $n \geq 0$ and $m \geq 0$, the tree $tree(\Xi)$ represented by Ξ is:



Definition 7. *A set of formulae Θ is saturated iff it satisfies:*

1. *If $A \vee B \in \Theta$ then $A \in \Theta$ and $B \in \Theta$.*
2. *If $A \wedge B \in \Theta$ then $A \in \Theta$ or $B \in \Theta$.*

Definition 8. *Given a tree T and a node $\Theta \in T$, a formula $\Box A \in \Theta$ ($\blacksquare A \in \Theta$) is realised iff there exists a $\circ$-child ($\bullet$-child) Γ of Θ in T with $A \in \Gamma$.*

6.1 Proof Search in DKt

Figure 7 gives a proof search strategy for **DKt**. The application of a rule deep inside a sequent can be viewed as focusing on a particular node of the tree. The rules of **DKt** can then be viewed as operations on the tree encoded in the sequent. In particular, Step 3 saturates a node locally, Step 4 appends new nodes to the tree, and Step 5 moves $\Diamond$ ($\blacklozenge$) prefixed formulae between neighbouring nodes.

Definition 9. *Given a tree T and a node $\Theta \in T$, we say Θ is propagated iff:*

$\Diamond_1$**:** *for every $\Diamond A \in \Theta$ and for every $\circ$-child Γ of Θ, we have $A \in \Gamma$*
$\blacklozenge_1$**:** *for every $\blacklozenge A \in \Theta$ and for every $\bullet$-child Γ of Θ, we have $A \in \Gamma$*
$\Diamond_2$**:** *for every $\bullet$-child Γ of Θ and for every $\Diamond A \in \Gamma$, we have $A \in \Theta$*
$\blacklozenge_2$**:** *for every $\circ$-child Γ of Θ and for every $\blacklozenge A \in \Gamma$, we have $A \in \Theta$*

The degree of a formula is the maximum number of nested modalities:

$$\begin{aligned} deg(p) &= 0 \\ deg(A \# B) &= max(deg(A), deg(B)) \text{ for } \# \in \{\wedge, \vee\} \\ deg(\# A) &= 1 + deg(A) \text{ for } \# \in \{\Box, \Diamond, \blacksquare, \blacklozenge\}. \end{aligned}$$

The degree of a set of formulae is the maximum degree over all its members. We write $sf(A)$ for the subformulae of A, and define the set of subformulae of a set Θ as $sf(\Theta) = \bigcup_{A \in \Theta} sf(A)$. For a sequent Ξ we define $sf(\Xi)$ as below:

$$\begin{aligned} \Xi &= \Theta, \circ\{\Gamma_1\}, \cdots, \circ\{\Gamma_n\}, \bullet\{\Delta_1\}, \cdots, \bullet\{\Delta_m\} \\ sf(\Xi) &= sf(\Theta) \cup sf(\Gamma_1) \cup \cdots \cup sf(\Gamma_n) \cup sf(\Delta_1) \cup \cdots \cup sf(\Delta_m). \end{aligned}$$

Theorem 9. *Function Prove terminates for any input sequent Ξ.*

6.2 Proof Search in DS4

Let $\mathbf{DS4}^-$ denote the system **DS4** minus the rules $\Diamond_1$, $\Diamond_2$, $\blacklozenge_1$, $\blacklozenge_2$.

Theorem 10. *For every Γ, we have $\vdash_{\mathbf{DS4}^-} \Gamma$ if and only if $\vdash_{\mathbf{DS4}} \Gamma$.*

Proof. $\Rightarrow$: obvious since every rule of $\mathbf{DS4}^-$ is a rule of **DS4**. $\Leftarrow$: by induction on the height of the proof of $\vdash_{\mathbf{DS4}} \Gamma$, using the admissibility of $\Diamond_1$, $\Diamond_2$, $\blacklozenge_1$, $\blacklozenge_2$.

We now modify the *Prove* function for proof search in **DS4**. The saturation and propagatation of $\Diamond$- and $\blacklozenge$-prefixed formulae need to cater for reflexivity and transitivity respectively. Moreover, a loop check and blocking on the creation of new nodes is required, since a naive approach leads to non-termination [9]. We implement the loop check by adding histories to our nodes (in our case in the form of tagged formulae), thus extending Heuerding's approach [9] to tense logic.

A *tagged formula* is a formula of the form A^*. For $\# \in \{\Box, \Diamond, \blacksquare, \blacklozenge\}$, we write $(\#\Gamma)^*$ to mean a set of tagged #-formulae. In the following, the nodes in our trees will consist of sets of formulae and tagged formulae. Note that we use tagged formulae for book-keeping only; tagged formulae are never principal in inference rule applications. Let $\mathbf{DS4}^*$ be $\mathbf{DS4}^-$ with the T_a, T_b, $\Box$, $\blacksquare$ rules replaced by the following. For simplicity, we write them directly as tree expansion rules:

- **T_a^1 (T_b^1):** If some node Θ contains an untagged $\blacklozenge A$ ($\Diamond A$), add A to Θ, tag $\blacklozenge A$ ($\Diamond A$) and untag all $\Box$ ($\blacksquare$) formulae.
- **T_a^2 (T_b^2):** If some Θ contains a tagged $\blacklozenge A$ ($\Diamond A$), add A to Θ.
- **$\Box A$:** If some node Θ contains an unrealised, untagged $\Box A$, create an $\circ$-child $\{(\Box\Gamma)^*, (\Diamond\Delta)^*, A\}$, where $\Box\Gamma$ are all the $\Box$ formulae in Θ, and $(\Diamond\Delta)^*$ are all the tagged $\Diamond$ formulae in Θ.
- **$\blacksquare A$:** If some node Θ contains an unrealised, untagged $\blacksquare A$, create a $\bullet$-child $\{(\blacksquare\Gamma)^*, (\blacklozenge\Delta)^*, A\}$, where $\blacksquare\Gamma$ are all the $\blacksquare$ formulae in Θ, and $(\blacklozenge\Delta)^*$ are all the tagged $\blacklozenge$ formulae in Θ.

The intuition of tagging is that $\Box$ ($\blacksquare$) formulae are only expanded once within each cycle of repeated $\Diamond$ ($\blacklozenge$) formulae. If an untagged $\Diamond$ ($\blacklozenge$) formula is encountered, rule T_a^1 (T_b^1) removes the tags from all tagged $\Box$ ($\blacksquare$) formulae so that they can be expanded again. Eventually all $\Diamond$ ($\blacklozenge$) formulae will be tagged, so the $\Box$ ($\blacksquare$) formulae will also remain tagged and the $\Box$ ($\blacksquare$) rules will be blocked.

Definition 10. *A set of formulae Θ is $S4$-saturated iff it is saturated and $\Diamond A \in \Theta$ or $\blacklozenge A \in \Theta$ implies $A \in \Theta$.*

Definition 11. *A node Θ in a given tree T is $S4$-propagated iff:*

4_a: *for every $\blacklozenge A \in \Theta$ and for every $\bullet$-child Γ of Θ, we have $\blacklozenge A \in \Gamma$*
4_b: *for every $\circ$-child Γ of Θ and for every $\blacklozenge A \in \Gamma$, we have $\blacklozenge A \in \Theta$*
4_c: *for every $\Diamond A \in \Theta$ and for every $\circ$-child Γ of Θ, we have $\Diamond A \in \Gamma$*
4_d: *for every $\bullet$-child Γ of Θ and for every $\Diamond A \in \Gamma$, we have $\Diamond A \in \Theta$*

Let *ProveS*4 be the function *Prove* from Figure 7, modified as follows:

1. Replace "saturated" with "$S4$-saturated" and add four sub-steps to Step 3 for $\Diamond$- and $\blacklozenge$-formulae based on the rules T^1_a, T^1_b, T^2_a and T^2_b.
2. Replace "propagated" with "$S4$-propagated" in Step 5, and use Definition 11 instead of Definition 9.

Lemma 8. *For every* **DS4****-derivation* Π*, for every sequent* $\Xi \in \Pi$*, the maximum number of consecutive* $\circ$*-edges in tree*(Ξ) *is* m^2*, where* $m = |sf(\Xi)|$*.*

Lemma 9. *For every* **DS4****-derivation* Π*, for every sequent* $\Xi \in \Pi$*, the maximum number of consecutive* $\bullet$*-edges in tree*(Ξ) *is* m^2*, where* $m = |sf(\Xi)|$*.*

Theorem 11. *Function ProveS*4 *terminates for any input sequent* Ξ*.*

Proof. Let $T = tree(\Xi)$. The argument for Steps 3 and Step 5 is similar for the proof of Theorem 9. We need show that the depth of T is bounded by the loop check side conditions on the rules T^1_a, T^1_b, $\Box$, $\blacksquare$.

For a contradiction, suppose there exists a T of infinite depth, i.e., T contains an infinite branch. By Lemmas 8 and 9, an infinite branch must contain an infinite number of alternations between sequences of $\circ$-labelled edges and $\bullet$-labelled edges. Since $\circ$-children are created by applications of the $\Box$-rule and $\bullet$-children are created by applications of the $\blacksquare$-rule, there must be an infinite number of alternating $\Box$ and $\blacksquare$ rule applications (with any other rule applications in between). However, every such alternation decreases the degree of the node by at least 1, since the $\Box$ ($\blacksquare$) rule removes the outer $\Box$ ($\blacksquare$) from the principal formula, and $\Diamond$ ($\blacklozenge$) formulae can only be propagated across $\circ$ ($\bullet$) edges. Thus an infinite number of alternating $\Box$ and $\blacksquare$ rules is impossible. Contradiction.

7 Related Work and Future Work

Bernardi [1] appears to be the first to have noticed the connection between deep inference and residuation in display logic in the context of categorial grammar, although they do not give an explicit proof of this correspondence. Brünnler [3,4] and Poggiolesi [15] have given deep inference calculi for the modal logic K and some extensions. Brünnler has recently shown that the deep-inference-based cut-elimination technique for K [3] can be extended to prove cut elimination for Kashima's deep inference calculus for Kt.[1] In his proof, a crucial step is a proof of the admissibility of a "deep" version of residuation:

$$\frac{\Sigma[\bullet\{\circ\{\Delta\}, \Gamma\}]}{\Sigma[\Delta, \bullet\{\Gamma\}]} \qquad \frac{\Sigma[\circ\{\bullet\{\Delta\}, \Gamma\}]}{\Sigma[\Delta, \circ\{\Gamma\}]}$$

It will be interesting to compare the direct proof of cut elimination in deep systems (without residuation) to the one in shallow system (with residuation).

[1] K. Brünnler. Personal communication.

Indrzejczak [11] and Trzesicki [16] have given cut-free sequent-like calculi for tense logic. In each such calculus there is a rule (or rules) which allow us to "return" to previously seen worlds when the rules are viewed from the perspective of counter-model construction. However, Trzesicki's calculus has a large degree of non-determinism and is therefore not suitable for proof search. In contrast, our system **DKt** and its extension to tense S4 admits a simple proof search strategy and termination argument. Indrzejczak's calculus is suitable for proof search but lacks a natural notion of a cut rule and cut-elimination. It is also possible to give proof calculi for many modal and tense logics using semantic methods such as labelled deduction [14] and graph calculi [5], but we prefer purely syntactic methods since they can potentially be applied to logics with more complicated semantics such as substructural logics.

The description logic community have already built extremely efficient theorem provers for Kt.S4 in its incarnation as ALCI with transitive roles [10], so our terminating calculus for Kt.S4 is not very exciting. However, Horrocks et. al. do not consider proof-theoretic issues such as cut-elimination.

It remains to be seen whether we can extend our results to the primitive extensions of modal tense logic in a systematic way, and also whether deep inference can be used to tame other display calculi with more complex binary residuation principles like those in substructural logics [1]. Another interesting direction is the addition of (first-order) quantifiers. An approach to this would be to consider quantifiers as modal operators, with appropriate display postulates, such as the ones developed in [18].

A simple Haskell implementation of **DKt** is available at:
`http://users.rsise.anu.edu.au/~linda/DKt.html`.

Acknowledgment. We thank the anonymous referees for their helpful comments on an earlier draft of the paper.

References

1. Areces, C., Bernardi, R.: Analyzing the core of categorial grammar. Journal of Logic, Language, and Information 13(2), 121–137 (2004)
2. Belnap, N.: Display logic. Journal of Philosophical Logic 11, 375–417 (1982)
3. Brünnler, K.: Deep sequent systems for modal logic. In: Governatori, G., et al. (eds.) Advances in Modal Logic 6, pp. 107–119. College Publications (2006)
4. Brünnler, K.: Deep sequents for modal logic (unpublished, 2007)
5. Castilho, M.A., Cerro, L.F.D., Gasquet, O., Herzig, A.: Modal tableaux with propagation rules and structural rules. Fundamenta Informaticae 32(3/4), 281–297 (1997)
6. Goré, R., Gaggles, Gentzen, Galois: How to display your favourite substructural logic. Logic Journal of the IGPL 6(5), 669–694 (1998)
7. Goré, R.: Substructural logics on display. LJIGPL 6(3), 451–504 (1998)
8. Goré, R., Postniece, L., Tiu, A.: Cut-elimination and proof-search for bi-intuitionistic logic using nested sequents. In: AiML. College Publications (2008)

9. Heuerding, A., Seyfried, M., Zimmermann, H.: Efficient loop-check for backward proof search in some non-classical propositional logics. In: Miglioli, P., Moscato, U., Ornaghi, M., Mundici, D. (eds.) TABLEAUX 1996. LNCS(LNAI), vol. 1071, pp. 210–225. Springer, Heidelberg (1996)
10. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for very expressive description logics. Logic Journal of the IGPL 8(3), 239–264 (2000)
11. Indrzejczak, A.: Multiple sequent calculus for tense logics. In: International Conference on Temporal Logic, Leipzig, pp. 93–104 (2000)
12. Kashima, R.: Cut-free sequent calculi for some tense logics. Studia Logica 53, 119–135 (1994)
13. Kracht, M.: Power and weakness of the modal display calculus. In: Wansing, H. (ed.) Proof Theory of Modal Logics, pp. 92–121. Kluwer, Dordrecht (1996)
14. Negri, S.: Proof analysis in modal logic. JPL 34(5–6), 507–544 (2005)
15. Poggiolesi, F.: The tree-hypersequent method for modal propositional logic. In: Trends in Logic: Towards Mathematical Philsophy, pp. 9–30 (2009)
16. Trzesicki, K.: Gentzen-style axiomatization of tense logic. Bulleting of the Section of Logic 13(2), 75–84 (1984)
17. Wansing, H.: Sequent calculi for normal modal proposisional logics. Journal of Logic and Computation 4(2), 125–142 (1994)
18. Wansing, H.: Displaying Modal Logic. Kluwer Academic Publishers, Dordrecht (1998)

Sound Global State Caching for *ALC* with Inverse Roles

Rajeev Goré[1] and Florian Widmann[2]

[1] Logic and Computation Group, The Australian National University
Canberra, ACT 0200, Australia
`Rajeev.Gore@anu.edu.au`

[2] Logic and Computation Group and NICTA⋆, The Australian National University
Canberra, ACT 0200, Australia
`Florian.Widmann@anu.edu.au`

Abstract. We give an optimal (EXPTIME), sound and complete tableau-based algorithm for deciding satisfiability with respect to a TBox in the logic *ALCI* using global state caching. Global state caching guarantees optimality and termination without dynamic blocking, but in the presence of inverse roles, the proofs of soundness and completeness become significantly harder. We have implemented the algorithm in OCaml, and our initial comparison with FaCT++ indicates that it is a promising method for checking satisfiability with respect to a TBox.

1 Introduction

Description logics are classical multi-modal logics with applications in knowledge representation and reasoning [1]. Most applications can be reduced to the problem of deciding whether a given concept is satisfiable with respect to a finite set of concepts called a TBox. This problem is known to be EXPTIME-complete for the most basic expressive description logic *ALC* (normal multi-modal logic K_n), and known to be NEXPTIME-complete for more expressive logics like *SHOIQ* [2].

The known optimal algorithms [1] for these decision problems are rarely used by practitioners because they are difficult to implement. Practitioners have instead implemented sub-optimal, typically tableau-based, algorithms which exhibit good average-case behaviour by utilising a vast array of optimisations like "back-jumping" and "lazy unfolding" to reduce the tableau search space [3].

Tableau calculi for description logics typically build and-trees of nodes where each node can be viewed as a set of concepts, and where the or-branching caused by disjunctions is conceptually handled by splitting one and-tree into several. An important optimisation is to "cache" previously seen tableau nodes when their status is either known to be, or can safely be assumed to be, satisfiable or unsatisfiable [4]. If the same node appears again then a (hopefully fast) "cache hit"

⋆ NICTA is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence program.

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 205–219, 2009.

gives us the answer without having to explore the node's subtree again. Caching unsatisfiable nodes is sound across different and-trees, but caching satisfiable nodes can only be done within the same and-tree.

Goré and Nguyen have recently given an optimal, sound and complete algorithm for deciding $\mathcal{ALC}$-satisfiability with respect to a TBox which globally caches all nodes, regardless of their status [5]. Their "global caching" algorithm never explores the same node twice, immediately giving an optimal and terminating procedure. The main difficulty is to prove that the method is sound and complete. Recent experimental work of Goré and Postniece [6] has shown that the method is competitive with the existing caching methods. Goré and Nguyen have extended their method to several extensions of $\mathcal{ALC}$ by using an analytic cut-rule [7]. It is doubtful if these extended methods will lead to practical implementations because blind use of analytic cut is considered "impractical" by practitioners in this field. It is therefore important to find direct methods for these extension which utilise global caching without recourse to analytic cut.

One such extension is $\mathcal{ALCI}$ which extends $\mathcal{ALC}$ with inverse roles (converse modalities). Inverse roles cause problems because a concept like $\langle r\rangle([r^-]\varphi_1 \sqcup [r^-]\varphi_2)$ in a node w causes the creation of an r-successor node v containing $[r^-]\varphi_1 \sqcup [r^-]\varphi_2$, which then demands that the parent node w contains φ_1 if we expand the first disjunct in v but demands that the parent node w contains φ_2 if we expand the second disjunct. Assuming we take the first disjunct, if the node w does not contain φ_1 then it is "incompatible" with its r-child v, so, in principle, we have to add φ_1 to w and re-process the new φ_1 in w. But the re-processing may well make w contain a new concept $[r^-]\psi$ which we then have to pass back to the parent of w, and so on. Moreover, if choosing the first disjunct leads to an inconsistency, we have to undo all additions caused by the insertion of φ_1 into w so that we can explore the second disjunct in its original context. Conceptually, this can be done by creating a copy of the and-tree for each choice-point. Practical algorithms use many optimisations to minimise the copying required to maintain such choice-points as well as "dynamic equality blocking" to avoid infinite loops caused by TBoxes and inverse roles [8].

In summary, these methods can only globally cache unsatisfiable nodes, must cleverly manage choice-points and require blocking to be dynamic. Actual implementations are often sub-optimal in terms of their worst-case complexity. Polynomial reductions from $\mathcal{ALCI}$ to $\mathcal{ALC}$ are also known [9].

Here, we give a sound, complete and cut-free method using "global state caching" for deciding satisfiability with respect to a TBox for $\mathcal{ALCI}$. As opposed to global caching, which guarantees that the same node is never explored twice, our method globally caches only state nodes, but this is sufficient to guarantee worst-case optimality. This restriction can be safely relaxed to globally cache certain non-state nodes as well, and we return to this issue in Section 5. Since our underlying data structure is a cyclic graph, the main technical difficulty is to prove soundness and completeness (but the proofs are omitted for lack of space).

We present our method as pseudo code rather than as traditional tableau rules because the treatment of special nodes and the procedure `update` gives our algorithm a non-local flavour. Thus a set of traditional local tableau "completion rules" would be cluttered by side-conditions to enforce the non-local aspects or would require a complicated strategy of rule applications.

Comparison of our OCaml implementation with FaCT++ shows that our method is a promising method for checking $ALCI$-satisfiability w.r.t. a TBox.

Section 2 contains the syntax and semantics of $ALCI$. Section 3 contains an overview of the algorithm, the detailed algorithm itself, and statements of the theorems on soundness, completeness and optimal complexity. Section 4 contains a fully worked example. Section 5 contains a brief description of the implementation and our initial experimental results, and concludes.

2 Syntax and Semantics

Definition 1. *Let $\mathcal{AR}$ and $\mathcal{AC}$ be disjoint and countably infinite sets of* role names *and* concept names*, respectively. The set $\mathcal{R}$ of all* role descriptions *and the set $\mathcal{C}$ of all* concept descriptions *are inductively defined as follows: $\mathcal{AR} \subseteq \mathcal{R}$; if r is in $\mathcal{R}$ then so is r^-; $\mathcal{AC} \subseteq \mathcal{C}$; if C and D are in $\mathcal{C}$ then so are $\neg C$, $C \sqcap D$, and $C \sqcup D$; if C is in $\mathcal{C}$ then so are $[r]C$ and $\langle r \rangle C$ for every $r \in \mathcal{R}$. A concept of the form $\langle r \rangle C$ and $[r]C$ is called a $\langle \cdot \rangle$- and $[\cdot]$-concept, respectively.*

Definition 2. *An* interpretation *$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a pair where $\Delta^{\mathcal{I}}$ is a non-empty set, the* domain *of $\mathcal{I}$, and $\Delta^{\mathcal{I}}$ is an* interpretation function *mapping every $A \in \mathcal{AC}$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and every $r \in \mathcal{AR}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. An interpretation function is inductively extended to concepts and roles as follows:*

$$
\begin{array}{lcl}
(\neg C)^{\mathcal{I}} & := & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & := & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} & := & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
([r]C)^{\mathcal{I}} & := & \{d \in \Delta^{\mathcal{I}} \mid \forall e.(d,e) \in r^{\mathcal{I}} \ \Rightarrow \ e \in C^{\mathcal{I}}\} \\
(\langle r \rangle C)^{\mathcal{I}} & := & \{d \in \Delta^{\mathcal{I}} \mid \exists e.(d,e) \in r^{\mathcal{I}} \ \& \ e \in C^{\mathcal{I}}\} \\
(r^-)^{\mathcal{I}} & := & \{(e,d) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\} \ .
\end{array}
$$

Definition 3. *An interpretation $\mathcal{I}$* satisfies *a (not necessarily finite) set of concepts $X \subseteq \mathcal{C}$ iff $\bigcap_{C \in X} C^{\mathcal{I}} \neq \emptyset$, and* validates *$X$ iff $\bigcap_{C \in X} C^{\mathcal{I}} = \Delta^{\mathcal{I}}$. A* TBox *$\mathcal{T} \subseteq \mathcal{C}$ is a finite set of concepts. A set $X \subseteq \mathcal{C}$ is* satisfiable *with respect to $\mathcal{T}$ iff there exists an interpretation which validates $\mathcal{T}$ and satisfies X.*

We extend the definitions to single concepts by interpreting them as singleton sets. Clearly $\mathcal{I}$ validates C iff it does not satisfy $\neg C$. Traditionally, a TBox is defined to be a finite set of terminological axioms of the form $C \sqsubseteq D$, where C and D are concepts, but the two definitions are equivalent.

Definition 4. *For a role $r \in \mathcal{R}$ we define $r^{\smile}$ as s if r is of the form s^-, and as r^- otherwise. A concept $C \in \mathcal{C}$ is in* negation normal form *if $\neg$ appears only directly before concept names and if all roles appearing in C are in $\mathcal{AR} \cup \{r^- \mid r \in \mathcal{AR}\}$. It is well known that, in $ALCI$, every concept C has a logically equivalent concept* $\mathrm{nnf}(C)$ *which is in negation normal form. A TBox $\mathcal{T}$ is in negation normal form if all concepts in $\mathcal{T}$ are in negation normal form.*

3 Algorithm, Soundness, Completeness and Termination

Given a TBox $\mathcal{T}$ and a concept D, both in negation normal form, our method searches for an interpretation which validates $\mathcal{T}$ and satisfies D by building an and-or graph. We start with a high level description of our algorithm.

3.1 Overview of the Algorithm

Recall that the standard strategy for rule applications in tableau algorithms is to apply the rules for decomposing $\sqcap$ and $\sqcup$ repeatedly until they are no longer applicable, giving a "saturated" node which contains only atoms, negated atoms, $\langle\cdot\rangle$-formulae and $[\cdot]$-formulae. Let us call such a "saturated" node a *state* and call the other nodes *prestates*. Thus the only rule applicable to a state x is the $\langle\cdot\rangle$-rule which creates a node containing $\{C\} \cup \{D \mid [r]D \in x\}$ for each $\langle r\rangle C \in x$. The standard strategy will now saturate any such child to obtain a state y, then apply the $\langle\cdot\rangle$-rule to y, and so on, until we find a contradiction, or find a repeated node, or find a state which contains no $\langle\cdot\rangle$-formulae. Let us call x the parent state of y since all intervening nodes are not states.

When inverse roles are present, we require that $\{E \mid [r^\smile]E \in y\} \subseteq x$, since y is then compatible with being an r-successor of x in the putative interpretation under construction. If some $[r^\smile]E \in y$ has $E \notin x$ then x is "too small", and must be enlarged into an alternative node x^+ by adding all such E. If any such E is a complex formula then the alternative node x^+ is not "saturated", and hence not a state. So we must saturate it using the $\sqcap/\sqcup$-rules until we reach a state. That is, a state x may conceptually be "replaced" by an alternative prestate x^+ which is an enlargement of x, and which may have to be saturated further in order to reach a state.

Our algorithm handles these "alternatives" by introducing a new type of node called a *special node*, introducing a new type of status called `toosmall`, allowing states to contain a field alt for storing these alternatives, and ensuring that a state always has a special node as its parent. When we need to replace a state x by its alternatives, the special node above x extracts these alternatives from the alt_x field and creates the required alternative nodes as explained next.

Referring to Fig. 1, suppose state x has an r-successor prestate ps_0, and further saturation of ps_0 leads to prestate ps_k, and an application of an $\sqcap/\sqcup$-rule to p_k will give a state y. Instead of directly creating y, we create a special node z which carries the same set of formulae as would y, and make z a child of ps_k. We now check whether z is compatible with its parent state x by checking whether $\{E \mid [r^\smile]E \in z\} \subseteq x$. If z is not compatible then we mark z as `toosmall`, and add $\{E \mid [r^\smile]E \in z\} \setminus x$ to the set of alternative sets contained in alt_x, without creating y, as shown in Fig. 1(a). If z is compatible with x, we create a state y if it does not already exist, and make the new/old y a child of z, as in Fig. 1(b).

Suppose that y is compatible with x and that either y is already `toosmall` or becomes so later because of some descendant state w of y. In either case, the attribute alt_y then contains a number of sets $y_1, y_2, \ldots, y_n$ (say), and the `toosmall` status of y is propagated to the special node z. In response, z will

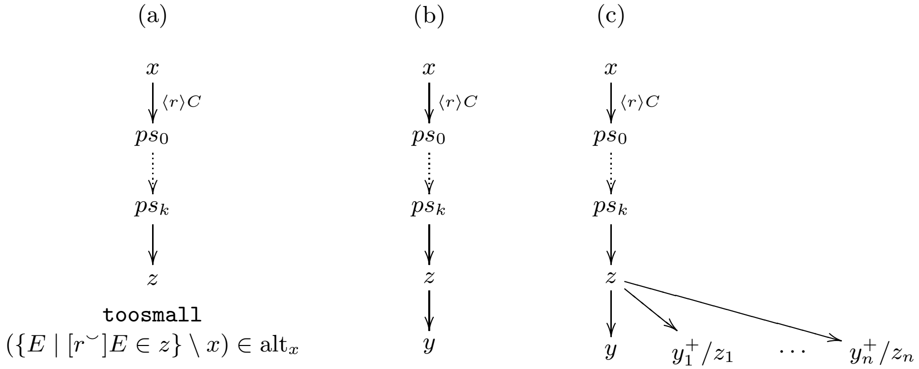


Fig. 1. The use of special node z to handle in/compatibility between states x and y. Scenario (a) occurs when x and y are incompatible. Scenario (b) occurs when x and y are compatible. Scenario (c) occurs when x and y are compatible, but y is `toosmall`.

create the alternatives $y_1^+, y_2^+, \ldots, y_n^+$ for y with $y_i^+ := y \cup y_i$. If y_i^+ is a state then our algorithm will create a special node z_i below z, and if z_i is compatible with x then y_i^+ will be created or retrieved and will become the child of z_i as in (b) else y_i^+ will not be created and z_i will be marked as `toosmall` as in (a). If y_i^+ is not a state then it will be created as a direct prestate child of z. Figure 1(c) captures this by using y_i^+/z_i to stand for either y_i^+ or z_i. Each of these new non-special nodes will eventually be expanded by our algorithm but now the "lapsed" special node z will be treated as a $\sqcup$-node.

3.2 The Algorithm

Our algorithm builds a graph G consisting of nodes and directed edges. We first explain the structure of G in more detail. In the rest of the paper, we use the notation $\mathscr{P}(Y)$ for the power set of Y and $x \in Y^?$ ($x \subseteq Y^?$) to indicate that x is either an element (a subset) of Y or undefined ("$\perp$").

Definition 5. *Each node $x \in G$ has six attributes belonging to it: $\Gamma_x \subseteq \mathcal{C}$, $\mathrm{alt}_x \subseteq \mathscr{P}(\mathcal{C})^?$, $\mathrm{pst}_x \in G^?$, $\mathrm{prl}_x \in \mathcal{R}^?$, $\mathrm{spl}_x \in (G \cup \{\mathrm{lsn}\})^?$, and $\mathrm{sts}_x \in \mathfrak{S}^?$ where $\mathfrak{S} := \{\texttt{unsat}, \texttt{sat}, \texttt{toosmall}, \texttt{open}\}$ and* lsn *is just a constant.*

Some attributes of a node $x \in G$ may be undefined initially. Once an attribute is defined in x, however, it will never become undefined again.

The attribute Γ_x of a node $x \in G$ contains the concepts that are assigned to x. It is set at the creation of x and is not changed afterwards. There may exist several nodes having the same set of concepts.

The attribute alt_x is defined (at the creation of x) if and only if x is a state. If defined it contains a set of sets of concepts. Each set of concepts can be seen as a way to extend Γ_x to form an *alternative node* for x. The set alt_x is initially empty but can grow as the algorithm proceeds.

The attributes pst_x and prl_x are defined for all nodes excepts states. They are set at the creation of x and are never changed. The attribute pst_x identifies the, as we will ensure, unique ancestor $p \in G$ of x such that p is a state and there is no other state between p and x in G. We call p the *parent state* of x. The creation of the child of p which lies between p and x was caused by a $\langle\cdot\rangle$-concept $\langle r\rangle C$ in Γ_p. The role r which we call the *parent role* of x is stored in prl_x.

The attribute spl_x is defined if and only if x is a *special node*. If defined, its value is either lsn or is the state that is the child of the special node. As explained in the overview, the special nature of special nodes can eventually lapse, after which they are treated as $\sqcup$-nodes: thus lsn stands for "lapsed special node".

The last attribute sts_x describes the *status* of x. It is initially undefined but becomes defined eventually during the algorithm. Its value may be modified several times. The value `unsat` indicates that the node is unsatisfiable. The value `sat` indicates that the node is satisfiable. The value `toosmall` indicates that the node is "useless" for building an interpretation because it does not contain some concepts that are required by inverse roles and $[\cdot]$-concepts. Hence, it is treated similarly to `unsat`. Finally, the value `open` indicates that it is currently not known whether or not the node is satisfiable.

Definition 6. *Let $x \in G$ be a node. We call x* unsat *iff it has* $\mathrm{sts}_x = \texttt{unsat}$*,* sat *iff it has* $\mathrm{sts}_x = \texttt{sat}$*,* too small *iff it has* $\mathrm{sts}_x = \texttt{toosmall}$*, and* open *iff it has* $\mathrm{sts}_x = \texttt{open}$*. A* path *$\pi$ in G is a finite or infinite sequence $x_0, x_1, x_2, \ldots$ of nodes in G such that x_{i+1} is a child of x_i for all x_i which have a successor in π.*

Next we comment on all procedures given in pseudocode.

Procedure `is-sat`$(D, \mathcal{T})$ is the main procedure which determines whether a concept $D \in \mathcal{C}$ is satisfiable w.r.t. a TBox $\mathcal{T}$, both in negation normal form. It first initialises G to the empty graph. We consider G as a global variable, so the other procedures have access to it. Then we create a dummy state which we call the *root node* and insert it in G. If we create a node, all attributes which are not explicitly given are undefined. The root node is inserted for technical reasons so that each node that is not a state has a parent state.

While there exists a node $x \in G$ whose status is undefined, we expand x as explained next. Since special nodes and nodes which contain a contradiction get their status in the invocation of `insert-node` which creates them, the following classifications do not contain such nodes.

If Γ_x contains a $\sqcap$-concept C whose immediate subconcepts are not in Γ_x, we call x a $\sqcap$-*node*, so we create a new set Γ' by adding C_1 and C_2 to Γ_x. Note $\Gamma' \supsetneq \Gamma_x$. We then invoke `insert-node` which creates a node with Γ' assigned to it and adds an edge from x to that node. Note that pst_x and prl_x are defined as x is not a state. After that we determine and set the status of x.

If x is not a $\sqcap$-node and Γ_x contains a $\sqcup$-concept C none of whose immediate subconcepts is in Γ_x, we call x a $\sqcup$-*node*. For each decomposition C_i we do the following: We create a new set Γ_i by adding C_i to Γ_x. Thus Γ_i is a strict superset of Γ_x. Then we invoke `insert-node` which creates a node with Γ_i assigned to it and add an edge from x to that node. Note that pst_x and prl_x must be defined as x is not a state. Finally, we determine and set the status of x.

Procedure. `is-sat`$(D, \mathcal{T})$ for testing whether D is satisfiable w.r.t. $\mathcal{T}$

```
Input: a concept $D \in \mathcal{C}$ and a TBox $\mathcal{T}$, both in negation normal form
Output: true iff $D$ is satisfiable w.r.t. $\mathcal{T}$

$G :=$ a new empty graph
let $s \in \mathcal{AR}$ be a dummy role name which does not occur in $D$ or $\mathcal{T}$
create new node rt with $\Gamma_{\text{rt}} := \{\langle s \rangle D\}$ and $\text{alt}_{\text{rt}} := \emptyset$
insert rt in $G$
while $\exists x \in G.\, \text{sts}_x = \bot$ do (* $x$ is not expanded yet *)
    if $\exists C \in \Gamma_x.\, C = C_1 \sqcap C_2$ & $\{C_1, C_2\} \not\subseteq \Gamma_x$ then (* $x$ is a $\sqcap$-node *)
        $\Gamma' := \Gamma_x \cup \{C_1, C_2\}$
        insert-node$(\Gamma', x, \text{pst}_x, \text{prl}_x)$
        $\text{sts}_x :=$ det-sts-or$(x)$
    else if $\exists C \in \Gamma_x.\, C = C_1 \sqcup C_2$ & $\{C_1, C_2\} \cap \Gamma_x = \emptyset$ then (* $x$ is a $\sqcup$-node *)
        for $i \longleftarrow 1$ to $2$ do
            $\Gamma_i := \Gamma_x \cup \{C_i\}$
            insert-node$(\Gamma_i, x, \text{pst}_x, \text{prl}_x)$
        $\text{sts}_x :=$ det-sts-or$(x)$
    else (* $x$ is a state *)
        let $\langle r_1 \rangle C_1, \cdots, \langle r_k \rangle C_k$ be all of the $\langle \cdot \rangle$-concepts in $\Gamma_x$
        for $i \longleftarrow 1$ to $k$ do
            $\Gamma_i := \{C_i\} \cup \{E \mid [r_i]E \in \Gamma_x\} \cup \mathcal{T}$
            insert-node$(\Gamma_i, x, x, r_i)$
        $\text{sts}_x :=$ det-sts-state$(x)$
    let $y_1, \ldots, y_k$ be all the parents of $x$
    for $i \longleftarrow 1$ to $k$ do update$(y_i)$
return $\text{sts}_{\text{rt}} \in \{$sat, open$\}$
```

If x is neither a $\sqcap$-node nor $\sqcup$-node, it must be fully saturated and hence a state. For each $\langle \cdot \rangle$-concept $\langle r_i \rangle C_i$ we do the following: We create a new set Γ_i containing C_i, all concepts in $\mathcal{T}$, and all concepts E such that $[r_i]E \in \Gamma_x$. Then we invoke `insert-node` which creates a node with Γ_i assigned to it and adds an edge from x to that node. We call this node the *successor* of $\langle r_i \rangle C_i$. Finally, we determine and set the status of x.

At the end of the while loop, we update the status of all parent nodes of x.

The procedure stops if all nodes in G have a defined status. It returns "satisfiable" iff the root node is either sat or open.

Procedure `insert-node`(Γ, x, p, r) nominally creates a node with Γ assigned to it and inserts an edge from x to that node. Due to the issues with inverse roles, however, the details are more complicated. We start by explaining the arguments of `insert-node` in more detail.

The node $x \in G$ invokes `insert-node` because it requires the existence of a node which has $\Gamma \subseteq \mathcal{C}$ assigned to it. The arguments $p \in G$ and $r \in \mathcal{R}$ are the parent state and the parent role of the new node, respectively. By inspecting the three invocations of `insert-node` in `is-sat`, it should not be hard to see that p and r are given the "right" values: if x is a $\sqcap$- or $\sqcup$-node then pst_x and prl_x

Procedure. `insert-node(`Γ, x, p, r`)` for inserting a node into the graph

```
Input: a set $\Gamma \subseteq \mathcal{C}$ containing the concepts of the new node; a node $x \in G$
       which invoked this procedure; the parent state $p \in G$ (in
       particular $\mathrm{alt}_p \neq \bot$); and the parent role $r \in \mathcal{R}$

if $\exists C \in \mathcal{C}.\, \{C, \mathrm{nnf}(\neg C)\} \subseteq \Gamma$ then (* contradiction found *)
    create new node $y$ with $\Gamma_y := \Gamma$, $\mathrm{pst}_y := p$, $\mathrm{prl}_y := r$, and $\mathrm{sts}_y :=$ unsat
    insert $y$ and edge $(x, y)$ in $G$
else if $\exists C \in \Gamma.\, C = C_1 \sqcap C_2$ or $C = C_1 \sqcup C_2$ then
    create new ($\sqcap$- or $\sqcup$-)node $y$ with $\Gamma_y := \Gamma$, $\mathrm{pst}_y := p$, and $\mathrm{prl}_y := r$
    insert $y$ and edge $(x, y)$ in $G$
else (* $\Gamma$ is fully saturated *)
    create new (special) node $z$ with $\Gamma_z := \Gamma$, $\mathrm{pst}_z := p$, and $\mathrm{prl}_z := r$
    $\Gamma_{\mathrm{alt}} := \{C \mid [r^{\smile}]C \in \Gamma\} \setminus \Gamma_p$
    if $\Gamma_{\mathrm{alt}} = \emptyset$ then (* $\Gamma$ is compatible with $p$ *)
        if $\exists y \in G.\, \mathrm{alt}_y \neq \bot \;\&\; \Gamma_y = \Gamma$ then (* state already exists in $G$ *)
            insert edge $(z, y)$ in $G$
            $\mathrm{spl}_z := y$
        else (* state is not in $G$ yet *)
            create new (state) node $y$ with $\Gamma_y := \Gamma$ and $\mathrm{alt}_y := \emptyset$
            insert $y$ and edge $(z, y)$ in $G$
            $\mathrm{spl}_z := y$
        $\mathrm{sts}_z :=$ det-sts-spl($z$)
    else (* $\Gamma$ is not compatible with $p$ *)
        if $\mathrm{sts}_p \in \{\bot, \texttt{open}\}$ then $\mathrm{alt}_p := \mathrm{alt}_p \cup \{\Gamma_{\mathrm{alt}}\}$
        $\mathrm{spl}_z := \mathrm{lsn}$
        $\mathrm{sts}_z =$ toosmall
    insert $z$ and edge $(x, z)$ in $G$
```

are just passed on; if x is a state then p is x itself and r is the role from the $\langle\cdot\rangle$-concept in x which requires the existence of the said node.

If Γ contains an immediate contradiction, we create a new node y which immediately becomes unsat and insert an edge from x to y. For the other cases, we assume implicitly that Γ does not contain an immediate contradiction.

If Γ contains a $\sqcap$- or $\sqcup$-concept which still has to be decomposed, we create a new $\sqcap$- or $\sqcup$-node y and insert an edge from x to y. Note that we create a new node even if there already exists a node in G which has Γ assigned to it; otherwise the parent state and the parent role of a node would not be unique.

If Γ is fully saturated, things become more interesting. In this case we first create a *special node* z, not because of the usual tableau rules, but to handle the "special" issue arising from inverse roles, as explained in the overview. Like $\sqcap$- and $\sqcup$-nodes, special nodes have a unique parent state and a unique parent role.

Next we determine the set Γ_{alt} of all concepts C such that $[r^{\smile}]C$ is in Γ but C is *not* in p. If there is no such concept we say that Γ is *compatible* with p. Note that incompatibilities can only arise because of inverse roles.

If Γ is compatible with p, we check whether some state y in G has Γ assigned to it. If such a state y already exists in G, we insert an edge from z to y; otherwise we create a new state y first and then insert an edge from z to y. Consequently, there is at most one state in G for every set of concepts explaining the term "global state caching" of the title. In both cases we flag z as special by defining $\mathrm{spl}_z := y$. Then we determine and set the status of z.

If Γ is not compatible with p, we cannot connect p to a state with Γ assigned to it as explained in the overview. Hence the intermediary z flags this by becoming too small. A node which is too small is treated similarly to an unsat node as both are useless for building an interpretation. That does not, however, mean that p is unsatisfiable; maybe it is just missing some concepts. We cannot extend Γ_p directly as this may have side-effects elsewhere; but to give p a clue as to what went wrong, we add Γ_{alt} to alt_p if p is still open or has an undefined status. The meaning is that if we create an alternative node for p by adding the concepts in Γ_{alt}, we might be more successful in building an interpretation. We flag z as special by defining $\mathrm{spl}_z := \mathrm{lsn}$, which in this case is just a dummy value which is not needed later.

Finally we put an edge from x to the special node z.

Note that if a special node z requires a state y which already exists in G *and is already known to be too small* then we must insert the alternative extensions of y immediately via `det-sts-spl`(z) to determine the status of z. Since such an alternative may itself be a special node, `insert-node` may recurse via `det-sts-spl`. This is why special nodes are the only nodes which get their status in the procedure `insert-node`, rather than in the outer procedure `is-sat`.

Procedure `det-sts-spl`(x) computes the status of a special node $x \in G$. By definition of a special node, the attribute spl_x is defined.

If spl_x is a node $y \in G$ then it must be a state and we do the following: If y is unsat or sat, the status is just propagated to x. If y is open or its status is not defined then x becomes open. The interesting case arises if y is too small, meaning that y is unsuitable for building an interpretation. Its attribute alt_y contains information on how to extend Γ_y in order to potentially fix the problem. So we do the following for every set $\Gamma \in \mathrm{alt}_y$: We create a new set by adding the concepts in Γ to Γ_y and then insert this alternative node of y in G and add an edge from x to it. It is easy to see that the new set is a strict superset of Γ_y. The alternative node does not have to be a state since it may contain $\sqcap$- and $\sqcup$-concepts, so it may require further saturation. Moreover, it is possible that it contains a contradiction or that its saturation leads to sets of concepts which are not compatible with the parent state of x. Hence we have to use `insert-node` to insert the alternative nodes. If one of the alternative nodes turns out to be "useful" for building an interpretation, it "replaces" the discarded y. Hence the special nature of x has "lapsed" and it behaves like a $\sqcup$-node from now on, so we set spl_z to lsn and determine its status by invoking `det-sts-or`.

If $\mathrm{spl}_x = \mathrm{lsn}$ then we know that x has already created the alternative nodes of the corresponding state and that it should behave like a $\sqcup$-node. Hence we invoke `det-sts-or` and pass on the result.

Procedure `det-sts-or`(x) computes the status of a $\sqcap$- or $\sqcup$-node $x \in G$. Note that for this task, a $\sqcap$-node can be seen as a $\sqcup$-node with exactly one child. If some child is sat then x is also sat. Otherwise, if there is at least one child that is open or has an undefined status, then x is still open. If none of the two cases apply, all children are unsat or too small. If there exists a child which is too small then x is too small. If all children are unsat then so is x. Note that, apart from the "extra" value `toosmall`, which is conceptually treated as `unsat`, the procedure captures the standard behaviour for tableaux.

Procedure `det-sts-state`(x) computes the status of a state $x \in G$. If one of the children of x is unsat then x must also be unsat. If some child y of x is too small then x must also be too small as y cannot be used for building an interpretation. If none of the children is unsat or too small, but some child is open or has an undefined status, then x is still open. If none of the other cases apply, all children must be sat, so x must be sat too. Again, apart from the "extra" value `toosmall`, which is conceptually treated as `unsat`, the procedure captures the standard behaviour for tableaux.

Procedure `update`(x) propagates status changes through G. It takes a node $x \in G$ and recomputes its status if the node is still open. If this new status differs from its old status stored in sts_x, it updates sts_x and invokes `update` recursively on all nodes whose status might be affected by this change. Note that a node that is open is either a special node or a $\sqcap/\sqcup$-node or a state.

We now list some facts which are useful in understanding the algorithm and which are needed in the (omitted) proofs of Theorems 8-10. These facts can be verified by inspection of the procedures in a rather straightforward way.

Proposition 7. *Let $x, y, z \in G$ be nodes.*

(i) *if `update`(x) is invoked then the status of x is defined;*
(ii) *if x and y are states with $\Gamma_x = \Gamma_y$ then $x = y$;*
(iii) *if x is a state then its parents are exactly the special nodes y with $\Gamma_y = \Gamma_x$;*
(iv) *if x is a state and $\Gamma \in \mathrm{alt}_x$ then $\Gamma \neq \emptyset$ and $\Gamma \cap \Gamma_x = \emptyset$;*
(v) *if x is a special node, it has at least one child iff $\{C \mid [\mathrm{prl}_x^{\smile}]C \in \Gamma_x\} \subseteq \Gamma_{\mathrm{pst}_x}$. In this case, one of its children is the state y with $\Gamma_y = \Gamma_x$.*
(vi) *if y is a child of x and neither of them are states then $\mathrm{pst}_x = \mathrm{pst}_y$ and $\mathrm{pst}_x = \mathrm{pst}_y$ and $\Gamma_x \subsetneqq \Gamma_y$;*

3.3 Soundness, Completeness, and Complexity

Let $D \in \mathcal{C}$ be a concept and $\mathcal{T}$ a TBox, both in negation normal form. Furthermore let G be the final graph with root node rt that was created by invoking `is-sat`$(D, \mathcal{T})$. Note that all nodes in G have a defined status. We define the size of a concept $C \in \mathcal{C}$ as the number of symbols in C. Let n be the sum of the sizes of all concepts in $X := \{D\} \cup \mathcal{T}$.

Theorem 8. *The algorithm terminates and runs in* EXPTIME *in n.*

Theorem 9. *If root node* rt *is sat or open then D is satisfiable w.r.t. $\mathcal{T}$.*

Theorem 10. *If* rt *is unsat or too small then D is not satisfiable w.r.t. $\mathcal{T}$.*

Procedure. `det-sts-spl(`x`)` for determining the status of a special node

```
Input: a special node $x \in G$ (i.e. $\mathrm{spl}_x \neq \bot$) with at least one child in $G$
Output: the new status of $x$

if $\mathrm{spl}_x \neq \mathrm{lsn}$ then (* $\mathrm{spl}_x$ is a child of $x$ *)
    $y := \mathrm{spl}_x$
    if $\mathrm{sts}_y =$ unsat then return unsat
    else if $\mathrm{sts}_y =$ sat then return sat
    else if $\mathrm{sts}_y \in \{\bot, \texttt{open}\}$ then return open
    else (* $y$ must be too small so create its alternatives *)
        foreach $\Gamma \in \mathrm{alt}_y$ do insert-node($\Gamma_y \cup \Gamma, x, \mathrm{pst}_x, \mathrm{prl}_x$)
        $\mathrm{spl}_x := \mathrm{lsn}$
        return det-sts-or($x$)
else return det-sts-or($x$)
```

Procedure. `det-sts-or(`x`)` for determining the status of a ⊓- or ⊔-node

```
Input: a ⊓- or ⊔-node $x \in G$
Output: the new status of $x$

let $y_1, \ldots, y_k \in G$ be all the children of $x$
if $\exists i \in \{1, \ldots, k\}.\ \mathrm{sts}_{y_i} =$ sat then return sat
else if $\exists i \in \{1, \ldots, k\}.\ \mathrm{sts}_{y_i} \in \{\bot, \texttt{open}\}$ then return open
else if $\exists i \in \{1, \ldots, k\}.\ \mathrm{sts}_{y_i} =$ toosmall then return toosmall
else return unsat; (* all children are unsatisfiable *)
```

Procedure. `det-sts-state(`x`)` for determining the status of a state

```
Input: a state $x \in G$
Output: the new status of $x$

let $y_1, \ldots, y_k \in G$ be all the children of $x$
if $\exists i \in \{1, \ldots, k\}.\ \mathrm{sts}_{y_i} =$ unsat then return unsat
else if $\exists i \in \{1, \ldots, k\}.\ \mathrm{sts}_{y_i} =$ toosmall then return toosmall
else if $\exists i \in \{1, \ldots, k\}.\ \mathrm{sts}_{y_i} \in \{\bot, \texttt{open}\}$ then return open
else return sat; (* all children are satisfiable *)
```

Procedure. `update(`x`)` for propagating the status of nodes

```
Input: a node $x \in G$ that has a defined status

if  $\mathrm{sts}_x =$ open then (* otherwise the status cannot change *)
    sts := if $\mathrm{spl}_x \neq \bot$ then det-sts-spl($x$)
              else if $\mathrm{alt}_x \neq \bot$ then det-sts-state($x$) else det-sts-or($x$)
    if $\mathrm{sts}_x \neq$ sts then
        $\mathrm{sts}_x :=$ sts
        let $z_1, \ldots, z_k$ be all the parents of $x$
        for $i \longleftarrow 1$ to $k$ do update($z_i$)
```

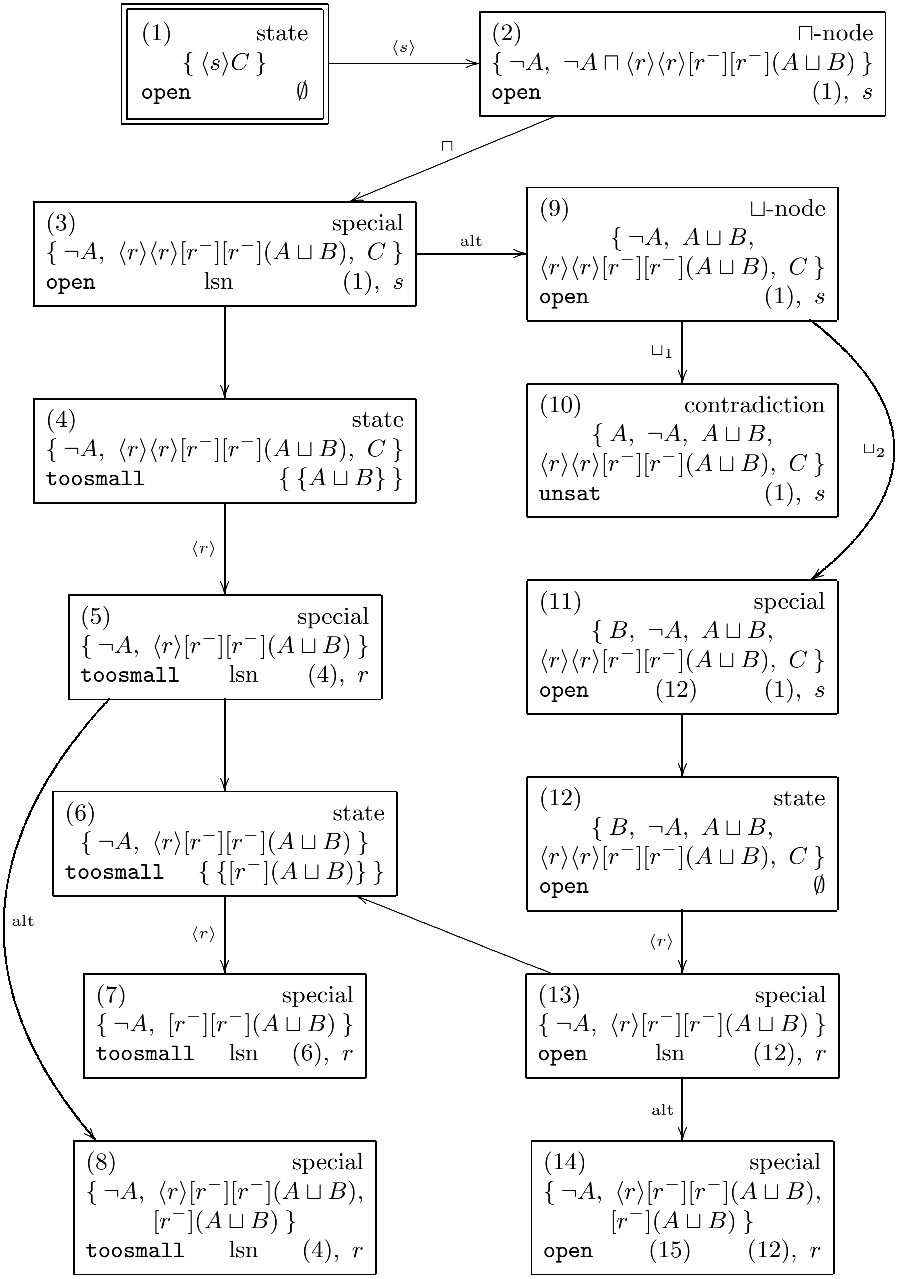

(1) state
$\{ \langle s \rangle C \}$
open $\emptyset$
$\langle s \rangle$
(2) $\sqcap$-node
$\{ \neg A, \ \neg A \sqcap \langle r \rangle \langle r \rangle [r^-][r^-](A \sqcup B) \}$
open (1), s
$\sqcap$
(3) special
$\{ \neg A, \ \langle r \rangle \langle r \rangle [r^-][r^-](A \sqcup B), \ C \}$
open lsn (1), s
alt
(9) $\sqcup$-node
$\{ \neg A, \ A \sqcup B,$
$\langle r \rangle \langle r \rangle [r^-][r^-](A \sqcup B), \ C \}$
open (1), s
$\sqcup_1$
$\sqcup_2$
(4) state
$\{ \neg A, \ \langle r \rangle \langle r \rangle [r^-][r^-](A \sqcup B), \ C \}$
toosmall $\{ \{A \sqcup B\} \}$
(10) contradiction
$\{ A, \ \neg A, \ A \sqcup B,$
$\langle r \rangle \langle r \rangle [r^-][r^-](A \sqcup B), \ C \}$
unsat (1), s
$\langle r \rangle$
(5) special
$\{ \neg A, \ \langle r \rangle [r^-][r^-](A \sqcup B) \}$
toosmall lsn (4), r
(11) special
$\{ B, \ \neg A, \ A \sqcup B,$
$\langle r \rangle \langle r \rangle [r^-][r^-](A \sqcup B), \ C \}$
open (12) (1), s
(6) state
$\{ \neg A, \ \langle r \rangle [r^-][r^-](A \sqcup B) \}$
toosmall $\{ \{[r^-](A \sqcup B)\} \}$
(12) state
$\{ B, \ \neg A, \ A \sqcup B,$
$\langle r \rangle \langle r \rangle [r^-][r^-](A \sqcup B), \ C \}$
open $\emptyset$
alt
$\langle r \rangle$
$\langle r \rangle$
(7) special
$\{ \neg A, \ [r^-][r^-](A \sqcup B) \}$
toosmall lsn (6), r
(13) special
$\{ \neg A, \ \langle r \rangle [r^-][r^-](A \sqcup B) \}$
open lsn (12), r
alt
(8) special
$\{ \neg A, \ \langle r \rangle [r^-][r^-](A \sqcup B),$
$[r^-](A \sqcup B) \}$
toosmall lsn (4), r
(14) special
$\{ \neg A, \ \langle r \rangle [r^-][r^-](A \sqcup B),$
$[r^-](A \sqcup B) \}$
open (15) (12), r

Fig. 2. An example: The graph just before processing node (17) (*cf.* Fig. 3)

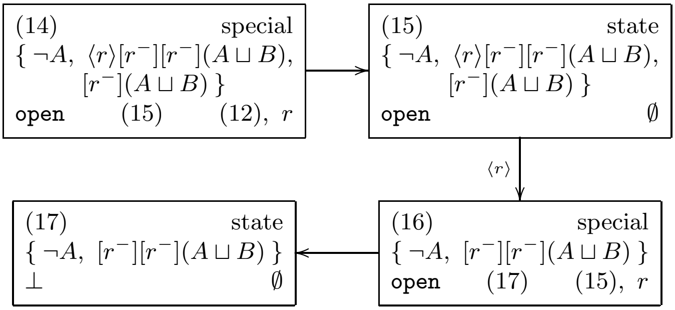

(14) special
$\{\neg A,\ \langle r\rangle[r^-][r^-](A\sqcup B),\ [r^-](A\sqcup B)\}$
open (15) (12), r
(15) state
$\{\neg A,\ \langle r\rangle[r^-][r^-](A\sqcup B),\ [r^-](A\sqcup B)\}$
open $\emptyset$
$\langle r\rangle$
(16) special
$\{\neg A,\ [r^-][r^-](A\sqcup B)\}$
open (17) (15), r
(17) state
$\{\neg A,\ [r^-][r^-](A\sqcup B)\}$
$\bot$ $\emptyset$

Fig. 3. An example: The graph just before processing node (17) (cont.)

4 A Fully Worked Example

Given the TBox $\mathcal{T} := \{\neg A\}$, the concept $C := \neg A \sqcap \langle r\rangle\langle r\rangle[r^-][r^-](A \sqcup B)$ is satisfiable w.r.t. $\mathcal{T}$, so our algorithm should say so. Note that $\mathcal{T}$ does not play a prominent role in this example and is only included for demonstration purposes.

Figure 2 and 3 show the graph just before processing node (17). The root is node (1) and Fig. 3 shows the subgraph rooted at node (14). The nodes are labelled in the order in which they are created. The bottom left corner of a node x contains sts_x. The bottom right corner contains alt_x if x is a state, and pst_x and prl_x if x is a $\sqcap/\sqcup$- or special node. If x is a special node then spl_x is given in the middle of the bottom line. We have not marked the principal concept which is decomposed in a node, since it is obvious. Arrows leaving states are labelled with a role r if an $\langle r\rangle$-concept caused the creation of the child. When a special node creates the alternative nodes, the edges that are created during this process are labelled with alt. The labelling of arrows leaving $\sqcap$- and $\sqcup$-nodes is obvious.

The creation and processing of nodes (1) and (2) is straightforward. As node (3) is compatible with state (1), state (4) is created and inserted as a child of node (3). Special node (5) and state (6) are handled similarly. Special node (7), however, is incompatible with state (6). Hence node (7) immediately becomes too small, and the set $\{[r^-](A \sqcup B)\}$ is added to the set of alternatives of state (6). Then state (6) becomes too small via `det-sts-state`. Its status change is propagated to special node (5) which now creates the alternative node (8) via `det-sts-spl`. Since node (8) is incompatible with state (4), it immediately becomes too small. Moreover the set $\{A \sqcup B\}$ is added to the set of alternatives of state (4). Because all children of node (5) are too small, it becomes too small as well via `det-sts-spl` and `det-sts-or`. This result is propagated to node (4) and (3). Similar to node (5), node (3) creates the alternative node (9).

The first child of node (9) contains a contradiction and becomes unsat immediately. Special node (11) and state (12) are handled similarly to node (3) and (4). Special node (13) is compatible with state (12) which contains no $[r^-]$-concepts. The state it requires is already in the graph as node (6), but it is too small, so node (13) immediately creates the alternative node (14). Nodes (8) and (14) are similar to each other but unlike node (8), node (14) *is* compatible with its parent state (12). So node (15) is created and inserted as its child. Node (16) is similar to node (7), but unlike node (7), it is compatible with its parent state (15). So state (17) is created and inserted as a child of node (16).

This is the moment captured by Fig. 2 and 3. Only node (17) remains unprocessed. Since it lacks $\langle\cdot\rangle$-concepts, it becomes sat via `det-sts-state` immediately. The result is propagated to all open nodes, including the root, which becomes sat. The algorithm returns that C is satisfiable w.r.t. $\mathcal{T}$.

5 Implementation, Experimental Results and Conclusion

Our algorithm is fairly detailed, so a naive implementation should be straightforward. But a well-engineered and sophisticated implementation requires more work so we address some aspects that do not show up in the algorithm.

There are some obvious optimisations. For example, the procedure can stop as soon as the root becomes sat, unsat or too small since we know that its status will not change again. Also we do not need to expand a node which has only parents who are all already sat, unsat or too small. Of course, if a new parent which is still open is linked to the node we might have to expand it after all.

We are free to choose any node to expand, but some nodes are obviously more "promising" than others. For example, as long as a $\sqcup$-node has an open child, it is not necessary to expand its other children. They require consideration only if this open child becomes unsat or too small. This can be implemented efficiently by having a decentralised queue which is distributed in the nodes.

One major issue is that our algorithm as given in this paper does the saturation phase for every state independently. That is, if two states differ only slightly, say one state has an additional concept name $A \in \mathcal{AC}$, the "same" saturation phase is done twice which seems unnecessary. However, we cannot unconditionally use the same saturation tree for both states; for example a concept $[r^-]\neg A$ in some node of the saturation tree might affect one state but not the other. Having said that, some improvements are possible. For example, we can cache and reuse unsat nodes in the saturation phase, but this is not possible for the other nodes. We can, however, reuse the same saturation tree for several states. That is, if we are about to recreate (parts of) a saturation tree, we do not create new nodes but allow the existing nodes to have several status flags, one for each state in whose saturation tree they appear. Caching unsat nodes in saturation trees is easy to implement, but the second improvement makes the algorithm significantly more complicated.

We have a fairly sophisticated implementation of our algorithm in OCaml. It uses some of the important optimisations like back-jumping, semantic branching, and local simplification [3], but does not implement many "at a world" optimisations and heuristics from SAT like reordering subconcepts of a $\sqcup$-concept.

We compared our implementation with FaCT++. Since FaCT++ handles $\mathcal{SROIQ}$ which is much more expressive than $\mathcal{ALCI}$, it is possible that this additional complexity slows FaCT++ down on $\mathcal{ALCI}$. On the other hand, FaCT++ is highly optimised. Our intention was not to make a thorough system comparison, but only to check whether our method is feasible. For the same reason, we do not give actual runtimes but just explain the qualitative results.

First, good benchmarks for *ALCI* do not seem to exist, and most knowledge bases use additional features like number restrictions, so we found it hard to compile a good set of test problems. We therefore used the `T98-sat` and `T98-kb` problems from the DL98 benchmarks (`http://dl.kr.org/dl98/comparison`). The only problems where FaCT++ did significantly better were the "pigeon hole" problems, which is not surprising since our prover does not include any SAT optimisations. In five problems, our prover showed significantly better results. For the remaining thirty problems, both provers were on par. We see these results only as a sanity check since these problems do not use inverse roles.

Second, we tested the provers on randomly generated *ALCI* concepts with varying sizes of *ALCI* TBoxes. For empty TBoxes, the concepts were not very difficult as both provers showed a linear increase in the size of the concepts and gave similar runtimes. As the TBoxes grew bigger, the performance of both provers started to degrade in a similar manner. Although randomly generated concepts are not good benchmarks, they sometimes serve a second purpose. We found that FaCT++ returned some incorrect results as confirmed by its authors.

Our method for *ALCI* is definitely promising and should be extended to more expressive logics to test whether it remains feasible. The extension to role hierarchies and transitive roles should not present difficulties, but the extension to include nominals and qualified number restrictions is not obvious to us.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
2. Horrocks, I., Sattler, U.: A tableau decision procedure for SHOIQ. Journal of Automated Reasoning 39(3), 249–276 (2007)
3. Horrocks, I., Patel-Schneider, P.F.: Optimizing description logic subsumption. Journal of Logic and Computation 9(3), 267–293 (1999)
4. Donini, F.M., Massacci, F.: EXPTIME tableaux for ALC. Artificial Intelligence 124(1), 87–138 (2000)
5. Goré, R., Nguyen, L.A.: EXPTIME tableaux for ALC using sound global caching. In: Proc. DL 2007. CEUR Workshop, vol. 250 (2007), `CEUR-WS.org`
6. Goré, R., Postniece, L.: An experimental evaluation of global caching for ALC (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 299–305. Springer, Heidelberg (2008)
7. Goré, R., Nguyen, L.A.: EXPTIME tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In: Olivetti, N. (ed.) TABLEAUX 2007. LNCS, vol. 4548, pp. 133–148. Springer, Heidelberg (2007)
8. Blackburn, P., Wolter, F., van Benthem, J. (eds.): Handbook of Modal Logic. Elsevier, Amsterdam (2006)
9. Ding, Y., Haarslev, V., Wu, J.: A new mapping from ALCI to ALC. In: Proc. DL-2007. CEUR Workshop Proceedings, vol. 250 (2007), `CEUR-WS.org`

A Tableau System for the Modal μ-Calculus

Natthapong Jungteerapanich

Laboratory for Foundations of Computer Science,
School of Informatics, University of Edinburgh, UK
n.jung@ed.ac.uk

Abstract. This paper presents a tableau system for determining satisfiability of modal μ-calculus formulas. The modal μ-calculus, which can be seen as an extension of modal logic with the least and greatest fixpoint operators, is a logic extensively studied in verification and has been shown to subsume many well-known temporal and modal logics including CTL, CTL*, and PDL. Concerning the satisfiability problem, the known methods in literature employ results from the theory of automata on infinite objects. The tableau system presented here provides an alternative solution which does not rely on automata theory. Since every tableau in the system is a finite tree structure (bounded by the size of the initial formula), this leads to a decision procedure for satisfiability and a small model property. The key features are the use of names to keep track of the unfolding of variables and the notion of name signatures used in the completeness proof.

1 Introduction

As a logic for specifying system properties, the modal μ-calculus is one of the most extensively studied. The logic was introduced by Kozen [4] as an extension of propositional modal logic with the least and greatest fixpoint operators. The fixpoint operators enable the logic to encode many well-known branching-time temporal logics and program logics including CTL, CTL*, and PDL; thus allow the logic to express the properties expressible in these latter logics, and many more. The incorporation of fixpoint operators, however, introduces difficulties when solving computational and logical problems. Our concern here is the satisfiability problem, i.e. to find a decision procedure which determines whether a formula is satisfiable. A partial solution was first given in [4] where a tableau method for checking satisfiability for a fragment of the logic, called *aconjunctive formulas*, was introduced. This also, at the same time, proved the small model property and the completeness of a deductive system for such fragment. However, for the full logic, the tableau method in that paper was insufficient. The decidability of the satisfiability problem for the full logic was first established in [5] where it was shown that the modal μ-calculus can be effectively encoded in the monadic second-order logic of n-successors (SnS). The satisfiability problem for SnS is known to be decidable [8] but is non-elementary.

A major milestone was made by Streett and Emerson [10], who introduced the notion of *well-founded pre-models* as a characterisation of models. The

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 220–234, 2009.

paper also suggested that the existence of a well-founded pre-model for the given formula can be checked by automata. Particularly, to show whether a formula is satisfiable, an infinite-tree automaton which accepts all well-founded tree pre-models for the formula is constructed; the formula is satisfiable iff the automaton accepts some tree (which can be seen as a tree model for the formula). A related method ([3], [6]) is to translate a formula into an equivalent alternating tree automaton, which is then checked for emptiness. As far as we know, these have been the only known effective methods to check satisfiability.

In this paper, we present a tableau system which can be directly used to check the satisfiability of a modal μ-calculus formula. A tableau in our tableau system is a finite tree structure whose size is bounded by (some function on) the size of the formula. A *successful* tableau can be seen as a model for the initial formula. The key is in the use of what we call *names* to keep track of the unfolding of variables in the tableaux. A formula labelling each node in a tableau is augmented with a sequences of names recording a (partial) history of unfolding of variables in such formula. The termination and success of a tableau are then determined from the recorded sequences of names. Since it is shown that every tableau is finite, the soundness and completeness of the tableau system entails the decidability of satisfiability and a small model property of the logic.

2 Modal μ-Calculus

Syntax. For convenience, we present the modal μ-calculus in *positive form*, where negation symbols can only appear next to proposition letters. Precisely, formulas in the modal μ-calculus are given by the following grammar:

$$\phi ::= P \mid \neg P \mid X \mid \phi \vee \phi \mid \phi \wedge \phi \mid \langle a\rangle\phi \mid [a]\phi \mid \mu X.\phi \mid \nu X.\phi$$

where P ranges over a countable set Prop of *proposition letters*, a over a countable set Act of *actions*, and X over a countable set Var of *variables*. It is well-known that every (closed) formula in the original syntax in [4] can be converted into an equivalent one in positive form.

We use σ to stand for either μ or ν. A *literal* is a proposition letter or its negation. Formulas of the form $\langle a\rangle\phi$, $[a]\phi$ and $\sigma X.\phi$ are called $\langle\cdot\rangle$*-formulas*, $[\cdot]$*-formulas*, and *fixpoint formulas*, respectively. An occurrence of a variable X in a formula is *free* iff its does not lie within the scope of σX; it is said to be *bound* otherwise. A *closed* formula is one without free occurrences of variables. $|\phi|$ denotes the length of ϕ.

Definition 1 (Well-named formula). *A formula ϕ is* well-named *iff, for each variable X, there is at most one operator of the form σX in ϕ and, if X occurs free in ϕ, no operator σX occurs in ϕ. Given a* well-named *formula ϕ and variable X, the* unique *fixpoint subformula $\sigma X.\psi$ of ϕ, if exists, is said to be* identified *by X. A μ-*variable *(resp. ν-variable) in ϕ is a variable which identifies a formula of the form $\mu X.\psi$ (resp. $\nu X.\psi$). A variable X is said to be* higher *(in ϕ) than variable Y iff the fixpoint formula identified by Y in ϕ is a proper subformula of the formula identified by X.*

Semantics. A *model* is a triple $\mathcal{M} = \langle\ M, \{R_a\}_{a \in \mathrm{Act}}, \mathcal{V}_{\mathrm{Prop}}\rangle$ where

- M is a set of *states*,
- R_a, for each action $a \in \mathrm{Act}$, is a binary relation on M, and
- $\mathcal{V}_{\mathrm{Prop}} : \mathrm{Prop} \to \wp(M)$, called a *propositional valuation.*

Suppose $\mathcal{M}$ is of above form. A *valuation* on $\mathcal{M}$ is a function $\mathcal{V} : \mathrm{Var} \to \wp(M)$. Given a valuation $\mathcal{V}$ on $\mathcal{M}$ and a formula ϕ, the set of states satisfying ϕ, denoted $\|\phi\|^{\mathcal{M}}_{\mathcal{V}}$, is given inductively as follows:

$$\begin{aligned}
\|P\|^{\mathcal{M}}_{\mathcal{V}} = \mathcal{V}_{\mathrm{Prop}}(P), \quad \|\neg P\|^{\mathcal{M}}_{\mathcal{V}} &= M - \mathcal{V}_{\mathrm{Prop}}(P),\\
\|X\|^{\mathcal{M}}_{\mathcal{V}} &= \mathcal{V}(X),\\
\|\phi_1 \vee \phi_2\|^{\mathcal{M}}_{\mathcal{V}} &= \|\phi_1\|^{\mathcal{M}}_{\mathcal{V}} \cup \|\phi_2\|^{\mathcal{M}}_{\mathcal{V}},\\
\|\phi_1 \wedge \phi_2\|^{\mathcal{M}}_{\mathcal{V}} &= \|\phi_1\|^{\mathcal{M}}_{\mathcal{V}} \cap \|\phi_2\|^{\mathcal{M}}_{\mathcal{V}},\\
\|\langle a\rangle\phi\|^{\mathcal{M}}_{\mathcal{V}} &= \{s \in M \mid \exists t.sR_at, t \in \|\phi\|^{\mathcal{M}}_{\mathcal{V}}\},\\
\|[a]\phi\|^{\mathcal{M}}_{\mathcal{V}} &= \{s \in M \mid \forall t.sR_at \to t \in \|\phi\|^{\mathcal{M}}_{\mathcal{V}}\},\\
\|\mu X.\phi\|^{\mathcal{M}}_{\mathcal{V}} &= \bigcap\{S \subseteq M \mid \|\phi\|^{\mathcal{M}}_{\mathcal{V}[X:=S]} \subseteq S\},\\
\|\nu X.\phi\|^{\mathcal{M}}_{\mathcal{V}} &= \bigcup\{S \subseteq M \mid S \subseteq \|\phi\|^{\mathcal{M}}_{\mathcal{V}[X:=S]}\}.
\end{aligned}$$

where $\mathcal{V}[X := S]$ is the valuation in which $\mathcal{V}[X := S](X) = S$ and $\mathcal{V}[X := S](Y) = \mathcal{V}(Y)$ for each variable Y other than X. For brevity, the superscript $\mathcal{M}$ is omitted if possible.

A formula ϕ is said to be *true* at state s in model $\mathcal{M}$ under valuation $\mathcal{V}$, written $\mathcal{M}, s \models_{\mathcal{V}} \phi$, iff $s \in \|\phi\|^{\mathcal{M}}_{\mathcal{V}}$. A formula is said to be *satisfiable* iff there is a model, a valuation, and a state satisfying it. A formula ϕ is said to be *valid*, written $\models \phi$, iff ϕ is true at every state in every model under any valuation. Two formulas ϕ, ψ are said to be *(semantically) equivalent* iff $\models \phi \leftrightarrow \psi$.

An *approximant* for $\sigma X.\psi$ is a formula of the form $\sigma^\alpha X.\psi$ (where α ranges over ordinals), whose semantics can be given as follows:

- $\|\mu^0 X.\psi\|_{\mathcal{V}} = \emptyset$ and $\|\nu^0 X.\psi\|_{\mathcal{V}} = M$,
- $\|\sigma^{\alpha+1} X.\psi\|_{\mathcal{V}} = \|\psi\|_{\mathcal{V}[X:=\|\sigma^\alpha X.\psi\|_{\mathcal{V}}]}$,
- $\|\mu^\lambda X.\psi\|_{\mathcal{V}} = \bigcup_{\alpha<\lambda} \|\mu^\alpha X.\psi\|_{\mathcal{V}}$ and $\|\nu^\lambda X.\psi\|_{\mathcal{V}} = \bigcap_{\alpha<\lambda} \|\nu^\alpha X.\psi\|_{\mathcal{V}}$,

where α denotes an ordinal and λ a limit ordinal. From Knaster-Tarski theorem, it is well-known that $\|\mu X.\psi\|_{\mathcal{V}} = \bigcup_\alpha \|\mu^\alpha X.\psi\|_{\mathcal{V}}$ and $\|\nu X.\psi\|_{\mathcal{V}} = \bigcap_\alpha \|\nu^\alpha X.\psi\|_{\mathcal{V}}$.

By renaming bound variables, every formula can be turned into an equivalent well-named formula. [7] and [6] define the notion of *guarded* formulas, and shows that every formula is semantically equivalent to a formula of such kind.

Definition 2 (Guarded formulas). *A formula ϕ is* guarded *iff, for each subformula $\sigma X.\psi$, each free occurrence of X in ψ lies within the scope of an occurrence of a modal operator $\langle\cdot\rangle$ or $[\cdot]$ in ψ.*

Lemma 1 ([7], [6]). *Every formula is semantically equivalent to a guarded one.*

Signatures. Streett and Emerson [10] introduced the notion of *signatures*, which has become an indispensable tool in the modal μ-calculus. The definitions given here are adapted from [11].

Let ϕ be a closed and well-named formula. Originally, a signature associates an *ordinal* to each μ-variable in ϕ. We extend the definition a bit by using elements in the class $\mathbb{O}^\infty = \{\alpha \mid \alpha \text{ an ordinal}\} \cup \{\infty\}$, and stipulate that $\alpha < \infty$ for each ordinal α. Fix a sequence $X_1, ..., X_m$ of all the variables in ϕ such that X_i higher than X_j implies $i < j$, and suppose $Z_1, ..., Z_n$ is the subsequence of the μ-variables in the former sequence. A *signature* is a sequence $\langle \alpha_1, ..., \alpha_n \rangle$ of elements in $\mathbb{O}^\infty$.

Let $\mathcal{M}$ be a model. Although ϕ is closed, a subformula of ϕ may contain free occurrences of variables. As in [11], we define the valuation which assigns their intended meanings to those variables. Precisely, the *valuation* $\mathcal{V}_{\mathcal{M},\phi}$ is defined to be $\mathcal{V}_m$, where $\mathcal{V}_0, ..., \mathcal{V}_m$ are given as follows:

- $\mathcal{V}_0(X) = \emptyset$ for all variables X;
- $\mathcal{V}_{i+1} = \mathcal{V}_i[X_{i+1} := \|\sigma_{i+1} X_{i+1}.\psi_{i+1}\|_{\mathcal{V}_i}]$.

Given a signature sig $= \langle \alpha_1, ..., \alpha_n \rangle$, the *relativised valuation* $\mathcal{V}^{\text{sig}}_{\mathcal{M},\phi}$ is defined to be $\mathcal{V}^{\text{sig}}_m$, where $\mathcal{V}^{\text{sig}}_0, ..., \mathcal{V}^{\text{sig}}_m$ are given as follows:

- $\mathcal{V}^{\text{sig}}_0(X) = \emptyset$ for all variables X;
- $\mathcal{V}^{\text{sig}}_{i+1} = \mathcal{V}^{\text{sig}}_i[X_{i+1} := \|\mu^{\alpha_j} Z_j.\psi\|_{\mathcal{V}^{\text{sig}}_i}]$, if X_{i+1} identifies $\mu Z_j.\psi$ and α_j is an ordinal;
- $\mathcal{V}^{\text{sig}}_{i+1} = \mathcal{V}^{\text{sig}}_i[X_{i+1} := \|\mu Z_j.\psi\|_{\mathcal{V}^{\text{sig}}_i}]$, if X_{i+1} identifies $\mu Z_j.\psi$ and $\alpha_j = \infty$;
- $\mathcal{V}^{\text{sig}}_{i+1} = \mathcal{V}^{\text{sig}}_i[X_{i+1} := \|\nu X_{i+1}.\psi\|_{\mathcal{V}^{\text{sig}}_i}]$, if X_{i+1} identifies $\nu X_{i+1}.\psi$.

For brevity, we write $\mathcal{M}, s \models \psi$ and $\mathcal{M}, s \models_{\text{sig}} \psi$ for $\mathcal{M}, s \models_{\mathcal{V}_{\mathcal{M},\phi}} \psi$ and $\mathcal{M}, s \models_{\mathcal{V}^{\text{sig}}_{\mathcal{M},\phi}} \psi$, respectively.

Lemma 2 ([10]). *For each subformula ψ of ϕ, if $\mathcal{M}, s \models \psi$ then there exists a signature* sig $= \langle \alpha_1, ..., \alpha_n \rangle$*, where each α_i is an ordinal, such that $\mathcal{M}, s \models_{\text{sig}} \psi$.*

3 Tableau System

We now describe the tableau system TS. Our presentation of tableaux has a close resemblance to the model-checking tableaux in [12]. To simplify matters, we only consider a tableau for a formula which is closed, guarded, and well-named. Obviously, every formula can be turned into a closed one without affecting satisfiability. As explained earlier, every formula is semantically equivalent to a guarded and well-named one. Thus, with some pre-processing, TS can be used to check satisfiability for any formula.

Suppose ϕ is a closed, guarded, and well-named formula. For definiteness, we fix a sequence $X_1, ..., X_m$ of all the variables in ϕ such that X_i higher than X_j implies $i < j$, and suppose $Z_1, ..., Z_n$ is the subsequence of the μ-variables. The tableau system TS employs extra symbols to keep track of the history of the unfoldings of μ-variables. For each μ-variable Z in ϕ, we assume a sequence $z^1, z^2, ...$ of distinct symbols, called *names* for Z. As we later show, the number of names required to build a tableau for ϕ is bound by the length of ϕ. For convenience, we use z, y, x or their scripted versions to denote names.

Goals. A *goal* in a tableau for ϕ is a sequent of the form $\mathit{\Theta} \vdash \mathit{\Gamma}$ where

- $\mathit{\Theta}$ is a sequence of *distinct* names (called a *global sequence*), and
- $\mathit{\Gamma}$ is a set of *augmented formulas* of the form ψ^ρ where ρ is a sequence of names from $\mathit{\Theta}$.

As shall be seen from the tableau rules, only the sequences ρ of special form will be used. Suppose ψ^ρ is an augmented formula in a goal $\mathit{\Theta} \vdash \mathit{\Gamma}$:

(1) ρ can be decomposed into $\rho(Z_1) \cdot ... \cdot \rho(Z_n)$ where each $\rho(Z_i)$ is a (possibly empty) sequence of names for Z_i.
(2) The ordering of names in ρ is compatible with that in $\mathit{\Theta}$.
(3) For any name z and formulas $\psi_1^{\rho_1}, \psi_2^{\rho_2}$ in $\mathit{\Gamma}$, if both ρ_1 and ρ_2 contain z, then the prefixes of ρ_1 and ρ_2 up to the occurrence of z are equal.

Names in a global sequence $\mathit{\Theta}$ are linearly ordered based on their positions in $\mathit{\Theta}$: for any names y and z in $\mathit{\Theta}$, $y <_{\mathit{\Theta}} z$ iff y occurs before z in $\mathit{\Theta}$. This extends to sequences of names in a lexicographical manner as follows: for any sequences ρ, ρ' of names in $\mathit{\Theta}$, $\rho \prec_{\mathit{\Theta}} \rho'$ iff, for some j, $\rho(j)$ and $\rho'(j)$ are names for the *same* variable and $\rho(i) = \rho'(i)$, $\rho(j) <_{\mathit{\Theta}} \rho'(j)$ for each $i < j$. Note that this latter ordering $\prec_{\mathit{\Theta}}$ is *not* total. For example, suppose $\mathit{\Theta} = z^1y^1z^2y^2y^3y^4$. Then $z^1y^1y^2y^4 \prec_{\mathit{\Theta}} z^1y^1y^3$, but z^1y^1 and $z^1z^2y^2$ are not comparable.

Given a sequence of names ρ and a variable X, $\rho \upharpoonright X$ denotes the sequence obtained from ρ by removing all the names for any variable appearing later than X in the sequence $X_1, ..., X_m$ assumed earlier. Similarly, for any number n, $\rho \upharpoonright n$ denotes the sequence of the first n names in ρ.

Tableau rules. A *tableau rule* is a rule of the form

$$\frac{\mathit{\Theta} \vdash \mathit{\Gamma}}{\mathit{\Theta}_1 \vdash \mathit{\Gamma}_1 \mid ... \mid \mathit{\Theta}_n \vdash \mathit{\Gamma}_n} \quad \mathcal{C}$$

where $n \geq 0$ and $\mathcal{C}$ is a *side condition.* The tableau rules of TS are given below. In the subgoals for rules Unfold_σ, Reset_z, and Thin, $\mathit{\Theta}'$ denotes the result of removing the names in $\mathit{\Theta}$ not appearing in any augmented formula in the subgoal. Similarly for $\mathit{\Theta}_i$ in the i-th subgoal of the rule $\mathsf{R}\langle\rangle$. This is to ensure that the names in $\mathit{\Theta}$ are precisely those associating some formula in the goal. See remarks below for further explanation.

$$\mathsf{R}\wedge: \quad \frac{\mathit{\Theta} \vdash (\psi_1 \wedge \psi_2)^\rho, \mathit{\Gamma}}{\mathit{\Theta} \vdash \psi_1^\rho, \psi_2^\rho, \mathit{\Gamma}} \qquad \mathsf{R}\vee: \quad \frac{\mathit{\Theta} \vdash (\psi_1 \vee \psi_2)^\rho, \mathit{\Gamma}}{\mathit{\Theta} \vdash \psi_i^\rho, \mathit{\Gamma}} \quad i \in \{1, 2\}$$

$$\mathsf{R}\sigma: \quad \frac{\mathit{\Theta} \vdash (\sigma X.\psi)^\rho, \mathit{\Gamma}}{\mathit{\Theta} \vdash X^\rho, \mathit{\Gamma}}$$

$$\mathsf{Unfold}_\mu: \frac{\mathit{\Theta} \vdash Z^\rho, \mathit{\Gamma}}{\mathit{\Theta}' \cdot z^i \vdash \psi^{(\rho\upharpoonright Z)\cdot z^i}, \mathit{\Gamma}}$$ where Z identifies $\mu Z.\psi$ and z^i is the *first* name for Z *not* occurring in $\mathit{\Theta}$.

$$\mathsf{Unfold}_\nu: \quad \frac{\mathit{\Theta} \vdash X^\rho, \mathit{\Gamma}}{\mathit{\Theta}' \vdash \psi^{\rho\upharpoonright X}, \mathit{\Gamma}},$$ where X identifies $\nu X.\psi$.

$$\mathsf{R}\langle\rangle : \frac{\Theta \vdash (\langle a_1\rangle\psi_1)^{\rho_1}, ..., (\langle a_n\rangle\psi_n)^{\rho_n}, \Gamma}{\Theta_1 \vdash \psi_1^{\rho_1}, \Gamma_{a_1} \mid ... \mid \Theta_n \vdash \psi_n^{\rho_n}, \Gamma_{a_n}}, n \geq 1,$$

where

- Γ contains only literals and $[\cdot]$-formulas, and
- for each action a, $\Gamma_a = \{\psi^\rho \mid ([a]\psi)^\rho \in \Gamma\}$.

$$\mathsf{Thin} : \quad \frac{\Theta \vdash \psi^\rho, \psi^{\rho'}, \Gamma}{\Theta' \vdash \psi^\rho, \Gamma},$$

where either $\rho \prec_\Theta \rho'$ or, for some μ-variable Z, $\rho' \upharpoonright Z$ is a *proper* prefix of $\rho \upharpoonright Z$.

$$\mathsf{Reset}_z : \quad \frac{\Theta \vdash \psi_1^{\rho\cdot z\cdot z_1\cdot\rho_1}, ..., \psi_n^{\rho\cdot z\cdot z_n\cdot\rho_n}, \Gamma}{\Theta' \vdash \psi_1^{\rho\cdot z}, ..., \psi_n^{\rho\cdot z}, \Gamma}, \quad n \geq 1,$$

where $z, z_1, ..., z_n$ are names for the same variable and z does *not* occur in Γ.

Remarks. In rule Unfold_μ, a new name for the μ-variable being unfolded is added to the global sequence. In order to bound the number of possible goals, we always choose the first name z^i for such μ-variable (i.e. one with the least i) not occurring in Θ.

The *thinning rule* Thin eliminates redundant formulas in the goal. It can be shown that, for any distinct formulas $\psi^\rho, \psi^{\rho'}$ in a goal, the condition specified in Thin (uniquely) chooses one of these formulas to keep.

Lemma 3. *Thin is applicable on any goal $\Theta \vdash \psi^\rho, \psi^{\rho'}, \Gamma$ where $\rho \neq \rho'$.*

Tableaux. A *tableau for ϕ* is a proof tree $\mathcal{T}$ whose root is labelled with the *initial goal* $\vdash \phi$ (i.e. the global sequence is empty). For each node u in $\mathcal{T}$ labelled by $\Theta \vdash \Gamma$, the goals labelling the children of u are determined by an application of a tableau rule, subject to the termination condition given below. To guarantee finiteness, when constructing a tableau it is required that rule Thin has the highest priority, following by rule Reset, i.e. rule Thin is always applied whenever possible, and in case Thin is not applicable, rule Reset_z, for any name z, is applied if possible.

Termination. A *terminal* in a tableau $\mathcal{T}$ is a node u labelled by $\Theta \vdash \Gamma$ such that *one* of the following conditions hold:

T1. Γ contains a complementary pair of literals (e.g. $P, \neg P$).

T2. Γ contains only literals and $[\cdot]$-formulas, but not a complementary pair of literals.

T3. u has a proper ancestor $v : \Theta \vdash \Gamma$, called the *companion* of u.

It is required that a terminal in $\mathcal{T}$ is a leaf (i.e. a terminal node is not expanded further).

Success. A *successful terminal* is a terminal u labelled by $\Theta \vdash \Gamma$ such that *one* of the following holds.

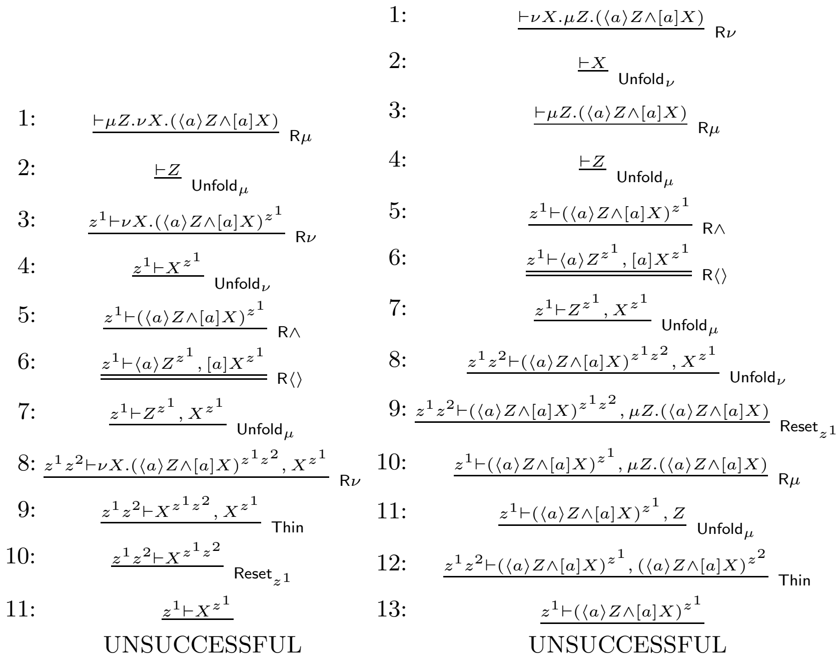
S1. u satisfies T2.

S2. u has companion v and, for each name z which occurs in every goal on the path from v to u, rule Reset_z is *not* applied between v and u.

A terminal is said to be *unsuccessful* otherwise.

A *successful tableau* $\mathcal{T}$ is a *finite* tableau all whose leaves are successful terminals.

Example 1. The formula $\mu Z.\nu X.(\langle a\rangle Z \wedge [a]X)$ is clearly unsatisfiable. As expected, every tableau for this formula is unsuccessful. One such tableau is shown in figure 1(a). The terminal node 11 is unsuccessful as it has node 4 as the companion, the name z^1 occurs in every goal from node 4 to node 11, and rule Reset_{z^1} is applied at node 10. Another example is the unsatisfiable formula $\nu X.\mu Z.(\langle a\rangle Z \wedge [a]X)$. An unsuccessful tableau for this formula is shown in figure 1(b).


1: $\vdash \mu Z.\nu X.(\langle a\rangle Z\wedge[a]X)$ Rμ
2: $\vdash Z$ Unfold$_\mu$
3: $z^1\vdash \nu X.(\langle a\rangle Z\wedge[a]X)^{z^1}$ Rν
4: $z^1\vdash X^{z^1}$ Unfold$_\nu$
5: $z^1\vdash (\langle a\rangle Z\wedge[a]X)^{z^1}$ R$\wedge$
6: $z^1\vdash \langle a\rangle Z^{z^1}, [a]X^{z^1}$ R$\langle\rangle$
7: $z^1\vdash Z^{z^1}, X^{z^1}$ Unfold$_\mu$
8: $z^1z^2\vdash \nu X.(\langle a\rangle Z\wedge[a]X)^{z^1z^2}, X^{z^1}$ Rν
9: $z^1z^2\vdash X^{z^1z^2}, X^{z^1}$ Thin
10: $z^1z^2\vdash X^{z^1z^2}$ Reset$_{z^1}$
11: $z^1\vdash X^{z^1}$
UNSUCCESSFUL
1: $\vdash \nu X.\mu Z.(\langle a\rangle Z\wedge[a]X)$ Rν
2: $\vdash X$ Unfold$_\nu$
3: $\vdash \mu Z.(\langle a\rangle Z\wedge[a]X)$ Rμ
4: $\vdash Z$ Unfold$_\mu$
5: $z^1\vdash (\langle a\rangle Z\wedge[a]X)^{z^1}$ R$\wedge$
6: $z^1\vdash \langle a\rangle Z^{z^1}, [a]X^{z^1}$ R$\langle\rangle$
7: $z^1\vdash Z^{z^1}, X^{z^1}$ Unfold$_\mu$
8: $z^1z^2\vdash (\langle a\rangle Z\wedge[a]X)^{z^1z^2}, X^{z^1}$ Unfold$_\nu$
9: $z^1z^2\vdash (\langle a\rangle Z\wedge[a]X)^{z^1z^2}, \mu Z.(\langle a\rangle Z\wedge[a]X)$ Reset$_{z^1}$
10: $z^1\vdash (\langle a\rangle Z\wedge[a]X)^{z^1}, \mu Z.(\langle a\rangle Z\wedge[a]X)$ Rμ
11: $z^1\vdash (\langle a\rangle Z\wedge[a]X)^{z^1}, Z$ Unfold$_\mu$
12: $z^1z^2\vdash (\langle a\rangle Z\wedge[a]X)^{z^1}, (\langle a\rangle Z\wedge[a]X)^{z^2}$ Thin
13: $z^1\vdash (\langle a\rangle Z\wedge[a]X)^{z^1}$
UNSUCCESSFUL

(a) (b)

Fig. 1. Unsuccessful tableaux for example 1

Example 2. Consider the satisfiable formula

$$(\nu X_1.(\mu Z.P \vee \langle a\rangle Z) \wedge \langle a\rangle X_1) \wedge (\mu Y.\nu X_2.(\neg P \wedge [a]X_2) \vee [a]Y)$$

A successful tableau for this formula is shown in figure 2.

$$
\begin{array}{llll}
1: & \underline{\vdash (\nu X_1.(\mu Z.P \vee \langle a\rangle Z) \wedge \langle a\rangle X_1) \wedge (\mu Y.\nu X_2.(\neg P \wedge [a]X_2) \vee [a]Y)}\ \mathsf{R}\wedge & & \\
2: & \underline{\vdash \nu X_1.(\mu Z.P \vee \langle a\rangle Z) \wedge \langle a\rangle X_1,\ \mu Y.\nu X_2.(\neg P \wedge [a]X_2) \vee [a]Y}\ \mathsf{R}\nu & & \\
3: & \underline{\vdash X_1,\ \mu Y.\nu X_2.(\neg P \wedge [a]X_2) \vee [a]Y}\ \mathsf{R}\mu & & \\
4: & \underline{\vdash X_1,\ Y}\ \mathsf{Unfold}_\mu & & \\
5: & \underline{y^1 \vdash X_1,\ (\nu X_2.(\neg P \wedge [a]X_2) \vee [a]Y)^{y^1}}\ \mathsf{R}\nu & & \\
6: & \underline{y^1 \vdash X_1,\ {X_2}^{y^1}}\ \mathsf{Unfold}_\nu & & \\
7: & \underline{y^1 \vdash X_1,\ ((\neg P \wedge [a]X_2) \vee [a]Y)^{y^1}}\ \mathsf{R}\vee & & \\
8: & \underline{y^1 \vdash X_1,\ (\neg P \wedge [a]X_2)^{y^1}}\ \mathsf{R}\wedge & & \\
9: & \underline{y^1 \vdash X_1,\ \neg P^{y^1},\ [a]{X_2}^{y^1}}\ \mathsf{Unfold}_\nu & & \\
10: & \underline{y^1 \vdash (\mu Z.P \vee \langle a\rangle Z) \wedge \langle a\rangle X_1,\ \neg P^{y^1},\ [a]{X_2}^{y^1}}\ \mathsf{R}\wedge & & \\
11: & \underline{y^1 \vdash \mu Z.P \vee \langle a\rangle Z,\ \langle a\rangle X_1,\ \neg P^{y^1},\ [a]{X_2}^{y^1}}\ \mathsf{R}\mu & & \\
12: & \underline{y^1 \vdash Z,\ \langle a\rangle X_1,\ \neg P^{y^1},\ [a]{X_2}^{y^1}}\ \mathsf{Unfold}_\mu & & \\
13: & \underline{y^1 z^1 \vdash (P \vee \langle a\rangle Z)^{z^1},\ \langle a\rangle X_1,\ \neg P^{y^1},\ [a]{X_2}^{y^1}}\ \mathsf{R}\vee & & \\
14: & \underline{\underline{y^1 z^1 \vdash \langle a\rangle Z^{z^1},\ \langle a\rangle X_1,\ \neg P^{y^1},\ [a]{X_2}^{y^1}}}\ \mathsf{R}\langle\rangle & & \\
15: & \underline{y^1 z^1 \vdash Z^{z^1},\ {X_2}^{y^1}}\ \mathsf{Unfold}_\mu & 15': & \underline{y^1 \vdash X_1,\ {X_2}^{y^1}} \\
 & & & \text{SUCCESSFUL} \\
16: & \underline{y^1 z^1 z^2 \vdash (P \vee \langle a\rangle Z)^{z^1 z^2},\ {X_2}^{y^1}}\ \mathsf{Reset}_{z^1} & & \\
17: & \underline{y^1 z^1 \vdash (P \vee \langle a\rangle Z)^{z^1},\ {X_2}^{y^1}}\ \mathsf{R}\vee & & \\
18: & \underline{y^1 z^1 \vdash P^{z^1},\ {X_2}^{y^1}}\ \mathsf{Unfold}_\nu & & \\
19: & \underline{y^1 z^1 \vdash P^{z^1},\ ((\neg P \wedge [a]X_2) \vee [a]Y)^{y^1}}\ \mathsf{R}\vee & & \\
20: & \underline{y^1 z^1 \vdash P^{z^1},\ [a]Y^{y^1}} & & \\
 & \text{SUCCESSFUL} & &
\end{array}
$$

Fig. 2. A successful tableau for example 2

Finiteness. It can be shown that every tableau is *finite*. This follows from the restriction that rules Thin and Reset are applied whenever possible and from the canonical choice of a new name introduced by rule Unfold_μ.

Lemma 4. *For each μ-variable Z, the names for Z occurring in each goal (in any tableau) are among $z^1, ..., z^{|\phi|}$.*

Proof. This property can be shown as an invariant when constructing a tableau for ϕ. In particular, we can show that when expanding a goal, if the supply of names in $\{z^1, ..., z^{|\phi|}\}$ runs out, then Thin or Reset must be applicable.

The previous lemma clearly implies that the number of possible goals in any tableau for ϕ is bounded. Let $|\mu\text{Var}(\phi)|$ denote the number of μ-variables in ϕ.

Lemma 5. *There are $2^{O(|\mu\text{Var}(\phi)||\phi|\log(|\phi|))}$ possible goals in a tableau for ϕ.*

Lemma 6. *Every tableau for ϕ is a finite tree of degree $O(|\phi|)$ and height $2^{O(|\mu\text{Var}(\phi)||\phi|\log(|\phi|))}$.*

Proof. The degree of a tableau cannot exceed the number of $\langle\cdot\rangle$-subformulas of ϕ, and hence is bounded by $O(|\phi|)$. By the previous lemma, a branch in a tableau cannot be longer than $2^{O(|\mu\text{Var}(\phi)||\phi|\log(|\phi|))}$.

4 Soundness

Suppose $\mathcal{T}$ is a *successful* tableau for a guarded and closed formula ϕ. $\mathcal{T}$ can be seen as a tree-with-backedges structure (where the backedges are from the leaves to their companions). A model for ϕ can be constructed by identifying each "modal node" as a state. A *modal node* is either a node where rule $\mathsf{R}\langle\rangle$ is applied or a leaf node which contains only $[\cdot]$-formulas and literals. For convenience, we use the letters s, t and their scripted versions to denote modal nodes. To define the transition relation, we need some extra notation. Suppose $\mathcal{T}$ is a tableau. For any nodes u, v in $\mathcal{T}$, we write $u \Rightarrow v$ when either v is a child of u or u is a leaf and v is its companion. For each modal node s, define the set

$[s] = \{u \mid$ there is a path $u = u_1 \Rightarrow ... \Rightarrow u_n = s$ $(n \geq 1)$ such that $\mathsf{R}\langle\rangle$ is *not* applied at u_i for each $i < n$ $\}$.

The *guardedness* of ϕ implies that for each node u there exists a *unique* modal node s such that $u \in [s]$.

Definition 3. *Suppose $\mathcal{T}$ is a tableau for a* guarded *formula. Define the* model corresponding to $\mathcal{T}$ *to be $\mathcal{M}_{\mathcal{T}} = \langle M, \{R_a\}_{a\in\text{Act}}, \mathcal{V}_{\text{Prop}}\rangle$ where*

- *M contains all modal nodes of $\mathcal{T}$,*
- *$sR_a t$ iff, for some node $u \in [t]$, a formula $\langle a\rangle\psi^\rho$ in s is reduced to ψ^ρ in u by rule $\mathsf{R}\langle\rangle$, and*
- *$\mathcal{V}_{\text{Prop}}(P) = \{s \in M \mid P^\rho$, for some ρ, is in the goal at s $\}$.*

It can be shown that $\mathcal{M}_{\mathcal{T}}$ is indeed a model for ϕ. To do so, we employ the notion of *trails* ([10], [7], [1]). A trail captures a sequence of reductions of formulas in a tableau. Precisely, given a tableau $\mathcal{T}$, a *trail* is a (finite or infinite) sequence $(u_1, \psi_1^{\rho_1}), (u_2, \psi_2^{\rho_2}), ...$ such that $u_1 \Rightarrow u_2 \Rightarrow ...$, each $\psi_i^{\rho_i}$ is in the goal at u_i and, for each $i \geq 1$, one of the following applies:

- The tableau rule applied at u_i reduces the formula $\psi_i^{\rho_i}$ in u_i to $\psi_{i+1}^{\rho_{i+1}}$ in u_{i+1}.
- The tableau rule applied at u_i does not reduce $\psi_i^{\rho_i}$ (thus $\psi_i^{\rho_i}$ is in u_{i+1}), and $\psi_{i+1}^{\rho_{i+1}} = \psi_i^{\rho_i}$.
- u_i is a terminal with u_{i+1} as its companion, and $\psi_{i+1}^{\rho_{i+1}} = \psi_i^{\rho_i}$.

(Note that in the case where Thin is applied to u_i labelled by $\Theta \vdash \psi^{\rho}, \psi^{\rho'}, \Gamma$ creating one successor u_{i+1} labelled by $\Theta' \vdash \psi^{\rho}, \Gamma$, both $(u_i, \psi^{\rho}), (u_{i+1}, \psi^{\rho})$ and $(u_i, \psi^{\rho'}), (u_{i+1}, \psi^{\rho})$ are counted as trails.)

An *unfolding* of a variable X in a trail is a subsequence of the form (u_i, X^{ρ}), $(u_{i+1}, \psi^{\rho'})$, i.e. rule Unfold is applied to X^{ρ} in u_i. X is said to be *unfolded* infinitely often in a trail iff there are infinitely many occurrences of unfoldings of X in the trail. In any infinite trail, there must be one or more variables unfolded infinitely often, and, particularly, the *highest* one. We call an infinite trail in which the highest variable unfolded infinitely often is a μ-variable a *μ-trail* [7]. For example, the following is a μ-trail in the tableau in figure 1(a):

$$(4, X^{z^1}) \to (5, (\langle a\rangle Z \wedge [a]X)^{z^1}) \to (6, \langle a\rangle Z^{z^1}) \to (7, Z^{z^1}) \to$$
$$(8, (\nu X.\langle a\rangle Z \wedge [a]X)^{z^1 z^2}) \to (9, X^{z^1 z^2}) \to (10, X^{z^1 z^2}) \to (11, X^{z^1}) \to (4, X^{z^1}) \to ...$$

The proof that $\mathcal{M}_{\mathcal{T}}$ is a model for ϕ is broken into two stages. First, we show that every successful tableau does *not* contain a μ-trail. Then show that, if $\mathcal{T}$ does *not* contain a μ-trail, $\mathcal{M}_{\mathcal{T}}$ is a model for ϕ.

Lemma 7. *Every successful tableau does* not *contain a μ-trail.*

Proof. Suppose $\mathcal{T}$ is a successful tableau which contains a μ-trail. Since the tableau is finite, such a μ-trail must contain a subtrail: $(u_1, \psi_1^{\rho_1}) \to (u_2, \psi_2^{\rho_2}) \to$ ... such that each $\psi_i^{\rho_i}$ occurs infinitely often in this subtrail. Suppose Z is the *highest* variable unfolded infinitely often in this subtrail. Observe that the rule Unfold_{μ} when applied to Z^{ρ} in a goal increases the length of $\rho \upharpoonright Z$. Since Z is unfolded infinitely often, the list $\rho_1 \upharpoonright Z$, $\rho_2 \upharpoonright Z$, ... does *not converge*, i.e. for each $i \geq 1$ there exists $i' > i$ such that $\rho_i \upharpoonright Z \neq \rho_{i'} \upharpoonright Z$. Let Y be the *highest* μ-variable which is higher than or equal to Z and such that the list $\rho_1 \upharpoonright Y$, $\rho_2 \upharpoonright Y$, ... does *not converge* (so Y could be Z). Consider the list $\rho_j \upharpoonright Y, \quad \rho_{j+1} \upharpoonright Y, \quad \rho_{j+2} \upharpoonright Y, \quad ....$ Let $\rho = \rho_k \upharpoonright Y$ (for some $k \geq j$) be an element in this list which has the least length, say n. It can be checked that, for each $i \geq j$, whatever tableau rule applied at u_i, $\rho_i \upharpoonright n \succeq_{\Theta_i} \rho_{i+1} \upharpoonright n$ (where Θ_i is the global sequence in u_i). Since ρ_j occurs infinitely often in the above list, it must be the case that $\rho_j \upharpoonright n = \rho_{j+1} \upharpoonright n = ...$, which implies that ρ is a prefix of each $\rho_j, \rho_{j+1}, ...$ Since the above list does not converge and ρ occurs infinitely often in the list (and is the shortest one so), either some variable higher than Y (and hence higher than Z) is unfolded infinitely often or rule Reset_x, where x is the last name in ρ, is applied infinitely often. The former cannot happen because Z is the highest variable unfolded infinitely often in the trail. Thus there must be a path $v, ..., u$ in $\mathcal{T}$, where u is a leaf and v is its companion, such that Reset_x is applied on the path and the name x occurs throughout the path. This contradicts the assumption that each leaf of $\mathcal{T}$ is successful.

Lemma 8. *If a tableau $\mathcal{T}$ for ϕ does not contain a μ-trail, $\mathcal{M}_{\mathcal{T}}$ satisfies ϕ.*

Proof. This lemma appears in various forms in literature; for instance, the tableau system in [7] and the *fundamental semantic theorem* in [1]. One way to prove this is to first define for each pair (u, ψ^{ρ}) where u is a node and ψ^{ρ} is in u, a *valuation* $\mathrm{Val}(u, \psi^{\rho})$:

- $\mathrm{Val}(u, \psi^\rho)(X) = \{s \in M \mid \exists v \in [s]$ s.t. there is a trail from (u, ψ^ρ) to $(v, X^{\rho'})$ *not* going through a variable higher than X $\}$.

Then prove a more general statement (by induction on ψ): for any state s and node $u \in [s]$, if ψ^ρ is in u, then $\mathcal{M}_\mathcal{T}, s \models_\mathcal{V} \psi$, where $\mathcal{V} = \mathrm{Val}(u, \psi^\rho)$.

Theorem 1 (Soundness). *Every closed, guarded, and well-named formula which has a successful tableau has a model in which the number of states is linear in the number of nodes in the tableau.*

5 Completeness

Every satisfiable (closed and well-named) formula ϕ has a successful tableau. The idea of the proof is to construct a successful tableau for ϕ by making choices which minimise a certain *measure* associated with the goals. The crucial part is to carefully define such a measure so that the constructed tableau guarantees to be successful.

Definition 4 (Name signatures). *A* name signature *is a function* $\eta : \mathsf{Names} \to \mathbb{O}^\infty$*, where* Names *is the set of all names. Name signatures are ordered with respect to a global sequence* Θ *as follows: suppose* $\Theta = z_1 ... z_n$*, define*

- $\eta \approx_\Theta \eta'$ *iff* $\eta(z_i) = \eta'(z_i)$ *for each* i*.*
- $\eta \prec_\Theta \eta'$ *iff* $\eta(z_j) < \eta'(z_j)$ *for some* j *and* $\eta(z_i) = \eta'(z_i)$ *for each* $i < j$*.*
- $\eta \preceq_\Theta \eta'$ *iff* $\eta \prec_\Theta \eta'$ *or* $\eta \approx_\Theta \eta'$*.*

Given a name signature η and a formula ψ^ρ, we can assign a signature for the μ-variables in ψ based on the names in ρ and the values given by η. Precisely, we define the signature $\eta_\rho = \langle \alpha_1, ..., \alpha_n \rangle$ as follows:

- $\alpha_i = \eta(z_i)$ if ρ contains a name for Z_i and z_i is the name for Z_i occurring *last* in ρ;
- $\alpha_i = \infty$, otherwise.

Definition 5. *A name signature* η *is considered* good *for* Γ *iff*

G1. for any sequence ρ *occurring in* Γ *and names* z^i, z^j *for the same variable, if* z^i *occurs before* z^j *in* ρ*, then* $\eta(z^i) > \eta(z^j)$*; and*
G2. there is a model $\mathcal{M}$ *and state* s *such that* $\mathcal{M}, s \models_{\eta_\rho} \psi$*, for each* $\psi^\rho \in \Gamma$*.*

By lemma 2, it is easy to see that every satisfiable set Γ has a good name signature.

Lemma 9. *For any goal* $\Theta \vdash \Gamma$*, if* Γ *is satisfiable, then there is a* good *name signature for* Γ*.*

Definition 6 (Signature of a goal). *For each goal* $\Theta \vdash \Gamma$ *where* Γ *is satisfiable, define* $\mathrm{Sig}(\Theta \vdash \Gamma)$ *to be the name signature* η *such that*

- η *is good for* Γ*,*
- $\eta \preceq_\Theta \eta'$ *for any good name signature* η' *for* Γ*, and*
- $\eta(z) = 0$ *for each name* z not *occurring in* Θ*.*

Clearly, a name signature η satisfying these conditions must be unique. The existence of such a name signature follows from the previous lemma. The following properties are essential in the completeness proof.

Lemma 10. *Below Θ' denotes the result of removing all the names in Θ not occurring in any augmented formula in the goal on the right hand side.*

(a) $\Gamma' \subseteq \Gamma$ *implies* $\mathrm{Sig}(\Theta \vdash \Gamma) \succeq_{\Theta'} \mathrm{Sig}(\Theta' \vdash \Gamma')$.
(b) $\mathrm{Sig}(\Theta \vdash (\psi_1 \wedge \psi_2)^\rho, \Gamma) \succeq_\Theta \mathrm{Sig}(\Theta \vdash \psi_1^\rho, \psi_2^\rho, \Gamma)$.
(c) $\mathrm{Sig}(\Theta \vdash (\psi_1 \vee \psi_2)^\rho, \Gamma) \succeq_\Theta \mathrm{Sig}(\Theta \vdash \psi_i^\rho, \Gamma)$ *for some* $i \in \{1, 2\}$.
(d) $\mathrm{Sig}(\Theta \vdash (\mu Z.\psi)^\rho, \Gamma) \succeq_\Theta \mathrm{Sig}(\Theta \vdash Z^\rho, \Gamma)$.
(e) $\mathrm{Sig}(\Theta \vdash (\nu X.\psi)^\rho, \Gamma) \succeq_\Theta \mathrm{Sig}(\Theta \vdash X^\rho, \Gamma)$.
(f) $\mathrm{Sig}(\Theta \vdash Z^\rho, \Gamma) \succeq_{\Theta'} \mathrm{Sig}(\Theta' \cdot z^i \vdash \psi^{(\rho \upharpoonright Z)\cdot z^i}, \Gamma)$ *where Z identifies $\mu Z.\psi$ and z^i is a name for Z* not *occurring in Θ.*
(g) $\mathrm{Sig}(\Theta \vdash X^\rho, \Gamma) \succeq_{\Theta'} \mathrm{Sig}(\Theta' \vdash \psi^{\rho \upharpoonright X}, \Gamma)$ *where X identifies $\nu X.\psi$.*
(h) $\mathrm{Sig}(\Theta \vdash (\langle a \rangle \psi)^\rho, \Gamma) \succeq_{\Theta'} \mathrm{Sig}(\Theta' \vdash \psi^\rho, \Gamma_a)$ *where* $\Gamma_a = \{\gamma^{\rho'} \mid ([a]\gamma)^{\rho'} \in \Gamma\}$.

Lemma 11. *Suppose $\Theta \vdash \psi_1^{\rho\cdot z\cdot z_1\cdot \rho_1}, ..., \psi_n^{\rho\cdot z\cdot z_n\cdot \rho_n}, \Gamma$ is a goal where $z, z_1, ..., z_n$ are names for the* same *variable, and z does* not *occur in Γ.*

$$\mathrm{Sig}(\Theta \vdash \psi_1^{\rho\cdot z\cdot z_1\cdot \rho_1}, ..., \psi_n^{\rho\cdot z\cdot z_n\cdot \rho_n}, \Gamma) \succ_{\Theta''} \mathrm{Sig}(\Theta' \vdash \psi_1^{\rho\cdot z}, ..., \psi_n^{\rho\cdot z}, \Gamma),$$

where Θ' is Θ with all the names not *occurring in the latter goal removed, and Θ'' is any prefix of Θ' which contains z.*

We are now ready to prove the completeness of TS. The tableau that we are constructing will have some uniformity which will later enable us to prove the small model property. A tableau is said to be *uniform* iff, for any pair of *non-terminal* nodes u, v with the same goal, the tableau rule applied at u is the same as the one applied at v, and the goals of the children of u are the same as those of the children of v.

Theorem 2 (Completeness). *Every satisfiable, closed, and well-named formula has a successful and uniform tableau.*

Proof. Suppose ϕ is a satisfiable, closed, and well-named formula. To construct a successful tableau for ϕ, we start with the smallest tableau $\mathcal{T}_0$ and subsequently expand it while making sure the set of formulas in each goal satisfiable. To guarantee uniformity, we assume a *selection rule* which, given a goal, specifies which formulas in the goal should be reduced first (giving priority to the formulas reducible via Thin or Reset). Suppose we have constructed $\mathcal{T}_0, ..., \mathcal{T}_i$. For each non-terminal leaf $u : \Theta \vdash \Gamma$ in $\mathcal{T}_i$, apply the tableau rule following to the assumed selection rule. We consider some interesting cases:

- $\Gamma = \underline{(\psi_1 \vee \psi_2)^\rho}, \Gamma'$. Rule $\mathsf{R}\vee$ can be applied to create either $\Theta \vdash \psi_1^\rho, \Gamma'$ or $\Theta \vdash \psi_2^\rho, \Gamma'$. By lemma 10(c), there is a *least* i such that ψ_i, Γ' is satisfiable and $\mathrm{Sig}(\Theta \vdash \Gamma) \succeq_\Theta \mathrm{Sig}(\Theta \vdash \psi_i^\rho, \Gamma')$. Apply $\mathsf{R}\vee$ to create the i-th subgoal.
- $\Gamma = \underline{Z^\rho}, \Gamma'$. Apply Unfold_μ to create the subgoal $\Theta' \cdot z^i \vdash \psi^{(\rho \upharpoonright Z)\cdot z^i}, \Gamma'$ where z^i is the first name for Z *not* occurring in Θ. By lemma 10(f), $\mathrm{Sig}(\Theta \vdash \Gamma) \succeq_{\Theta'} \mathrm{Sig}(\Theta' \cdot z^i \vdash \psi^{(\rho \upharpoonright Z)\cdot z^i}, \Gamma')$.

- $\Gamma = \underline{\psi_1^{\rho\cdot z\cdot z_1\cdot\rho_1}, ..., \psi_n^{\rho\cdot z\cdot z_n\cdot\rho_n}}, \Gamma'$ where $z, z_1, ..., z_n$ are names for the same variable, and z does *not* occur in Γ'. Apply Reset_z to create the subgoal $\Theta' \vdash \psi_1^{\rho\cdot z}, ..., \psi_n^{\rho\cdot z}, \Gamma'$. By lemma 11, $\mathrm{Sig}(\Theta \vdash \Gamma) \succ_{\Theta''} \mathrm{Sig}(\Theta' \vdash \psi_1^{\rho\cdot z}, ..., \psi_n^{\rho\cdot z}, \Gamma)$, for any prefix Θ'' of Θ' containing z.

In other cases, by lemma 10, for each created subgoal $\Theta' \vdash \Gamma'$, $\mathrm{Sig}(\Theta \vdash \Gamma) \succeq_{\Theta'} \mathrm{Sig}(\Theta' \vdash \Gamma')$. The construction must terminate at some tableau $\mathcal{T}'$ all whose leaves are terminal. Since each goal in $\mathcal{T}'$ is satisfiable, all the leaves which contain only literals and $[\cdot]$-formulas are successful. Other leaves in $\mathcal{T}'$ are also successful. Suppose $u_1 : \Theta_1 \vdash \Gamma_1, ..., u_n : \Theta_n \vdash \Gamma_n$ is the path to a terminal u_n from its companion u_1 (hence $\Theta_1 = \Theta_n$ and $\Gamma_1 = \Gamma_n$). Assume that u_n is unsuccessful. Thus there is some name z such that z occurs in each Θ_i and Reset_z is applied at some u_j, $1 \leq j < n$. Suppose $\Theta = z_1...z_k$, where $z_k = z$, is the prefix of Θ_1 up to the occurrence of z. Since $\Theta_1 = \Theta_n$, each z_i must also occur throughout the path, for if z_i is removed at some point, z_i cannot occur before z_k in Θ_n. Similarly, no name other than $z_1, ..., z_{k-1}$ may occur before z_k in each Θ_i on the path. This means that Θ is a prefix of each Θ_i. It follows from the construction that $\mathrm{Sig}(\Theta_1 \vdash \Gamma_1) \succeq_\Theta ... \succeq_\Theta \mathrm{Sig}(\Theta_n \vdash \Gamma_n)$. Since Reset_z is applied at u_j, by lemma 11, $\mathrm{Sig}(\Theta_j \vdash \Gamma_j) \succ_\Theta \mathrm{Sig}(\Theta_{j+1} \vdash \Gamma_{j+1})$. This is impossible because $\Theta_1 = \Theta_n$ and $\Gamma_1 = \Gamma_n$. Therefore u_n must be successful.

6 Applications

Since every tableau in TS is bounded in size, the soundness and completeness of TS imply a small model property and the decidability of the satisfiability problem. The complexity of the small model property and the satisfiability problem can also be obtained from the bound on tableaux.

Theorem 3. *Every satisfiable guarded formula ϕ has a finite model with $2^{O(|\mu\mathrm{Var}(\phi)||\phi|\log(|\phi|))}$ states.*

Proof. Assume w.l.o.g. that ϕ is closed and well-named. By completeness, ϕ has a successful and uniform tableau $\mathcal{T}$. From soundness, the model $\mathcal{M}_\mathcal{T}$ satisfies ϕ. Since $\mathcal{T}$ is uniform, $\mathcal{M}_\mathcal{T}$ can be turned into a small model by identifying all the states (i.e. modal nodes) with the same goal in $\mathcal{T}$. In particular, it is easy to show that, for any states s, t in $\mathcal{M}_\mathcal{T}$, if the goals at s and t in $\mathcal{T}$ are the same, s and t are bisimilar in $\mathcal{M}_\mathcal{T}$. By taking the bisimulation quotient on $\mathcal{M}_\mathcal{T}$ [1], we obtain a model for ϕ with $2^{O(|\mu\mathrm{Var}(\phi)||\phi|\log(|\phi|))}$ states.

Theorem 4. *The satisfiability problem for guarded formulas is in NEXPTIME.*

Proof. Suppose ϕ is a guarded formula. Assume w.l.o.g. that ϕ is closed and well-named. We construct a nondeterministic algorithm which determines whether ϕ has a successful tableau. As in the completeness proof, we assume a *selection rule* which, given a goal, specifies which formulas in the goal should be reduced first. Construct a *game graph* $\mathcal{G} = \langle V, E\rangle$ such that V contains all possible goals in tableaux for ϕ and $(\Theta \vdash \Gamma, \Theta' \vdash \Gamma') \in E$ iff the goal $\Theta \vdash \Gamma$ can be reduced to $\Theta' \vdash \Gamma'$ according to the selection rule. By lemma 5, $|V|$ is bounded by $2^{O(|\phi||\mu\mathrm{Var}(\phi)|\log(|\phi|))}$. A *choice node* for player I (II) is a goal where, according to the selection rule, $\mathsf{R}\vee$ (resp., $\mathsf{R}\langle\rangle$) is to be applied. A *play* is a *finite path* $\pi = \Theta_0 \vdash \Gamma_0, ..., \Theta_n \vdash \Gamma_n$ in the graph which is a branch in some maximal tableau for ϕ. Player I *wins* play π if the last node in π is a successful terminal. It is easy to see that player I has a *memoryless winning strategy* iff ϕ has a uniform successful tableau under the assumed selection rule. A nondeterministic algorithm first guesses a memoryless strategy for player I and then checks whether it is winning. The latter task can be carried out (deterministically) in time $O(|\phi||\mu\mathrm{Var}(\phi)||V|)$. Thus the algorithm nondeterministically determines whether ϕ is satisfiable in time $2^{O(|\phi||\mu\mathrm{Var}(\phi)|\log(|\phi|))}$.

Note that the algorithm given above in not optimal. It is known that the satisfiability problem for the modal μ-calculus is EXPTIME-complete ([2],[6],[1]). We believe that there is a *deterministic* algorithm which finds a successful tableau for ϕ in exponential time.

7 Conclusion and Related Work

The tableau system TS is closely related to the automata-theoretic method for checking satisfiability. As outlined in [10], an automaton recognising the tree models (of some pre-determined degree) of the given formula ϕ is constructed as a product of a *local automaton*, which is a tree automaton whose states are set of subformulas of ϕ, and a *checking automaton*, which is an infinite-word automaton checking that no "bad trail" exists on each branch of the tree. The checking automaton can be constructed from the complement of a simpler automaton (which recognises a branch containing a bad trail). If Safra's complementation method [9] is used, the states of the checking automaton will be Safra's trees. As a result, each state of the product automaton will have a tree structure. A goal in a tableau can be seen as a compact representation of such tree structure (To see this, suppose $\Theta \vdash \psi_1^{\rho_1}, ..., \psi_n^{\rho_n}$ is a goal. Let T be the tree whose nodes are the prefix closure of $\{\rho_1, ..., \rho_n\}$, and for each node ρ, $T(\rho)$ is labelled by the set of formulas $\psi_i^{\rho_i}$ where $\rho_i = \rho$). Rule Reset and the success condition of TS are both inspired by Safra's construction. Despite this connection, it is interesting to see that the soundness and completeness of the tableau system can be shown independent of the results from automata theory. More importantly, it is quite surprising that a simple form of measures, i.e. *name signatures*, is sufficient in guiding the construction of a successful tableau. We are investigating whether name signatures have applications elsewhere, e.g. in performing model surgery or in related automata theory.

What we present in this paper is a simple, yet powerful, tableau system for satisfiability. The nicest feature is that every tableau is a finite tree structure. As a result, we are able to derive both a small model property and a decision procedure for satisfiability. However, there is still much room for improvement. First, the guardedness assumption can be relaxed. The problem with unguarded formulas is that, in a tableau for such formula, we may be able to keep unfolding a fixpoint formula indefinitely without ever applying rule $\mathsf{R}\langle\rangle$. This can be solved by recording extra information into each formula which determines whether such formula is derived from an unfolding of a variable without rule $\mathsf{R}\langle\rangle$ applied in-between. We opt not to incorporate this mechanism into TS so that the presentation of the tableau system is as clear and simple as possible. But by doing so, the small model theorem and the decision procedure in the previous section apply to any formula. Secondly, the bound in the small model theorem obtained can still be improved. One way is to let the μ-variables in ϕ of the same alternation depth share the names. By doing so, we should be able to obtain a slightly better bound: $2^{O(k|\phi|\log(|\phi|))}$, where k is the alternation depth of ϕ. Finally, we hope that the tableau system presented in this paper will be useful for proving other properties of the logic; for instance, the completeness of Kozen's axiomatisation [4], [13].

Acknowledgements. I wish to thank my supervisor, Prof. Colin Stirling, for comments and guidance on the result in this paper.

References

1. Bradfield, J., Stirling, C.: Modal Mu-Calculi. In: Handbook of Modal Logic, pp. 721–756. Elsevier, Amsterdam (2007)
2. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs. In: Proc. 29th FOCS, pp. 328–337 (1988)
3. Emerson, E.A., Jutla, C., Sistla, A.P.: On model-checking for fragments of μ-calculus. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 385–396. Springer, Heidelberg (1993)
4. Kozen, D.: Results on the propositional mu-calculus. Theoret. Comput. Sci. 27, 333–354 (1983)
5. Kozen, D., Parikh, R.: A decision procedure for the propositional μ-calculus. In: Clarke, E., Kozen, D. (eds.) Logic of Programs 1983. LNCS, vol. 164, pp. 313–325. Springer, Heidelberg (1984)
6. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. Journal of ACM 47, 312–360 (2000)
7. Niwiński, D., Walukiewicz, I.: Games for the μ-calculus. Theoret. Comput. Sci. 163, 99–116 (1997)
8. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Tran. AMS 141, 1–35 (1969)
9. Safra, S.: On the complexity of ω-automata. In: Proc. 29th FOCS, pp. 319–327 (1988)
10. Streett, R.S., Emerson, E.A.: An automata theoretic decision procedure for the propositional mu-calculus. Information and Computation 81, 249–264 (1989)
11. Stirling, C.: Modal and Temporal Properties of Processes. Springer, Heidelberg (2000)
12. Stirling, C., Walker, D.: Local model checking in the modal mu-calculus. Theoret. Comput. Sci. 89, 161–177 (1991)
13. Walukiewicz, I.: Completeness of Kozen's axiomatization of the propositional μ-calculus. Information and Computation 157, 142–182 (2000)

Terminating Tableaux for Graded Hybrid Logic with Global Modalities and Role Hierarchies

Mark Kaminski, Sigurd Schneider, and Gert Smolka

Saarland University, Saarbrücken, Germany

Abstract. We present a terminating tableau calculus for graded hybrid logic with global modalities, reflexivity, transitivity and role hierarchies. Termination of the system is achieved through pattern-based blocking. Previous approaches to related logics all rely on chain-based blocking. Besides being conceptually simple and suitable for efficient implementation, the pattern-based approach gives us a NExpTime complexity bound for the decision procedure.

1 Introduction

Graded modal logic [1] is a powerful generalization of basic modal logic. Most prominently, graded modalities are used in description logics, rich modal languages tailored for knowledge representation that have a wide range of practical applications [2]. Graded modal logic allows to constrain the number of accessible states satisfying a certain property. So, the modal formula $\Diamond_n p$ is true in a state x if x has at least $n + 1$ successors satisfying p. Analogously to ordinary modal logic, graded modal logic can be extended by nominals [3]. The resulting language, graded hybrid logic, can be extended further by adding global modalities [4], which allow to specify properties that are to hold in all states.

Role hierarchies were first studied by Horrocks [5] in the context of description logics. Using inclusion assertions of the form $r \sqsubseteq r'$, one can specify that the role (relation) r is contained in the role r'. Role hierarchies are of particular interest when considered together with transitivity assertions for roles [6,7]. The description logic $\mathcal{SHOQ}$ [8] combines the expressive means provided by nominals, graded modalities, role hierarchies and transitive roles.

We present a terminating tableau calculus for graded multimodal logic extended by nominals, global modalities, reflexive and transitive roles, and role hierarchies. The modal language under consideration in the present work is equivalent to $\mathcal{SHOQ}$ extended by reflexive roles and a universal role, both extensions also being known from $\mathcal{SROIQ}$ [9].

The most important difference of our approach to existing calculi for $\mathcal{SHOQ}$ and stronger logics [8,10,9] is the technique used to achieve termination of the tableau construction. The established tableau algorithms all rely on modifications of Kripke's chain-based blocking technique [11]. Chain-based blocking assumes a precedence order on the nominals (also known as nodes or prefixes) of a tableau branch, and prevents processing of nominals that are subsumed by

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 235–249, 2009.

preceding nominals. In the simplest case, the precedence order is chosen to be the ancestor relation among nominals (ancestor blocking). In general, however, it may be any order that contains the ancestor relation (anywhere blocking [12,13]). Ancestor blocking gives an exponential bound on the length of ancestor chains, resulting in a double exponential bound on the size of tableau branches. Depending on the choice of the precedence order, anywhere blocking can lower this bound to a single exponential. However, the size bound on tableau branches does not seem to translate easily to a complexity bound for the decision procedures in [8,10,9] ([8,10] show a 2-NExpTime bound, while [9] leaves complexity open). We feel that the main difficulty in obtaining better complexity bounds is the algorithms being non-cumulative.

A tableau system is called cumulative if its rules never update or delete formulas. In contrast to most systems in the literature, calculi devised for description logics are often not cumulative. Cumulative calculi are easier to present than non-cumulative systems and are usually more amenable to analysis.

Unlike [8,10,9], our calculus is cumulative. Cumulativity of the calculus in the presence of nominals is achieved following [14] by representing equality constraints via an equivalence relation on nominals. Termination of our system is achieved through pattern-based blocking [15,14]. Pattern-based blocking is conceptually simpler than chain-based techniques in that it does not need an order on the nominals, and seems promising as it comes to efficient implementation [16]. Pattern-based blocking provides an exponential bound on the size of tableau branches and on the number of tableau rule applications for a single branch. Thus it limits the complexity of the associated decision procedure to NExpTime. To deal with graded modalities, we extend the blocking conditions in [15,14], preserving the exponential size bound on the tableau branches.

We begin by presenting a calculus for graded hybrid logic with global modalities. We argue that the blocking conditions used in [15,14] are insufficient in the presence of graded modalities. We extend pattern-based blocking to account for the increased expressive power and argue the completeness and termination of the resulting calculus. In the second part of the paper, we extend our calculus further by allowing reflexivity, transitivity and inclusion assertions. It turns out that in the presence of inclusion assertions, the blocking condition used for the basic calculus needs to be extended once again.

2 Graded Hybrid Logic with Global Modalities and Role Inclusion

Following [17,14], we represent modal logic in simple type theory (see [18]). This way we can make use of a rich syntactic and semantic framework and modal logic does not appear as an isolated formal system. We start with two base types B and S. The interpretation of B is fixed and consists of two truth values. The interpretation of S is a nonempty set whose elements are called *worlds* or *states*. Given two types σ and τ, the *functional type* $\sigma\tau$ is interpreted as the set of all total functions from the interpretation of σ to the interpretation of τ. We write $\sigma_1\sigma_2\sigma_3$ for $\sigma_1(\sigma_2\sigma_3)$.

We employ three kinds of variables: *Nominal variables* x, y, z of type S, *propositional variables* p, q of type SB, and *role variables* r of type SSB. Nominal variables are called *nominals* for short, and role variables are called *roles*. We assume there are infinitely many nominals. We use the logical constants

$$\bot, \top : \mathrm{B} \qquad \neg : \mathrm{BB} \qquad \vee, \wedge, \rightarrow : \mathrm{BBB} \qquad \doteq : \mathrm{SSB} \qquad \exists, \forall : (\mathrm{SB})\mathrm{B}$$

Terms are defined as usual. We write st for applications, $\lambda x.s$ for abstractions, and $s_1 s_2 s_3$ for $(s_1 s_2) s_3$. We also use infix notation, e.g., $s \wedge t$ for $(\wedge)st$.

Terms of type B are called *formulas*. We employ some common notational conventions: $\exists x.s$ for $\exists(\lambda x.s)$, $\forall x.s$ for $\forall(\lambda x.s)$, and $x \dot{\neq} y$ for $\neg(x \doteq y)$.

For every $n \in \mathbb{N}$ we define a constant $D_n : \mathrm{S} \ldots \mathrm{SB}$ as follows:

$$D_n \ := \ \lambda x_1 \ldots \lambda x_n . \bigwedge_{1 \leq i < j \leq n} x_i \dot{\neq} x_j$$

Without loss of generality, we assume a strict total order $\prec$ on the nominals. Given a set of nominals X of cardinality $n \geq 1$, we write DX for $D_n x_1 \ldots x_n$ where $X = \{x_1, \ldots, x_n\}$ and $x_i \prec x_{i+1}$ for $1 \leq i < n$. We write $\bar{D}X$ for $\neg DX$. Formulas of the form DX and $\bar{D}X$ are called *distinctness constraints* on X. Note that for two distinct variables x, y, $\bar{D}\{x, y\}$ reduces to $x \doteq y$.

Moreover, we use the following constants:

$$\begin{array}{ll} \sqsubseteq : (\mathrm{SSB})(\mathrm{SSB})\mathrm{B} & r_1 \sqsubseteq r_2 \ = \ \forall xy.r_1 xy \rightarrow r_2 xy \\ R : (\mathrm{SSB})\mathrm{B} & Rr \ = \ \forall x.rxx \\ T : (\mathrm{SSB})\mathrm{B} & Tr \ = \ \forall xyz.rxy \wedge ryz \rightarrow rxz \end{array}$$

To the right of each constant is an equation defining its semantics. We call formulas of the form $r \sqsubseteq r'$ *(role) inclusion assertions.* Formulas Rr and Tr are called *reflexivity* and *transitivity assertions*, respectively.

We write $\exists X.s$ for $\exists x_1 \ldots x_n.s$ if $|X| = n$ and $X = \{x_1, \ldots, x_n\}$. The *modal constants* are then defined as follows:

$$\begin{array}{rlrcl} \dot{\neg} : & (\mathrm{SB})\mathrm{SB} & \dot{\neg} px & = & \neg(px) \\ \dot{\wedge} : & (\mathrm{SB})(\mathrm{SB})\mathrm{SB} & (p \mathbin{\dot{\wedge}} q)x & = & px \wedge qx \\ \dot{\vee} : & (\mathrm{SB})(\mathrm{SB})\mathrm{SB} & (p \mathbin{\dot{\vee}} q)x & = & px \vee qx \\ \langle \text{-} \rangle_n : & (\mathrm{SSB})(\mathrm{SB})\mathrm{SB} & \langle r \rangle_n px & = & \exists Y.\, DY \wedge (\bigwedge_{y \in Y} rxy \wedge py) \\ [\text{-}]_n : & (\mathrm{SSB})(\mathrm{SB})\mathrm{SB} & [r]_n px & = & \forall Y.\, (\bigwedge_{y \in Y} rxy) \rightarrow \bar{D}Y \vee \bigvee_{y \in Y} py \\ E_n : & (\mathrm{SB})\mathrm{SB} & E_n px & = & \exists Y.\, DY \wedge \bigwedge_{y \in Y} py \\ A_n : & (\mathrm{SB})\mathrm{SB} & A_n px & = & \forall Y.\, \bar{D}Y \vee \bigvee_{y \in Y} py \\ \dot{\_} : & \mathrm{SSB} & \dot{x}y & = & x \doteq y \end{array}$$

where $|Y| = n + 1$ in all equations

Intuitively, the semantics of the graded modal operators is as follows:

$E_n p$: There are at least $n+1$ states satisfying p.
$A_n p$: All states but possibly n exceptions satisfy p.
$\langle r\rangle_n p$: There are at least $n+1$ r-successors satisfying p.
$[r]_n p$: All r-successors but possibly n exceptions satisfy p.

In accordance with the usual modal intuition, "formulas" of modal logic are seen as predicates of type SB denoting sets of states. They can be represented as *modal expressions* according to the following grammar:

$$t \ ::= \ p \mid \dot{x} \mid \dot{\neg} t \mid t \mathbin{\dot\wedge} t \mid t \mathbin{\dot\vee} t \mid \langle r\rangle_n t \mid [r]_n t \mid E_n t \mid A_n t$$

As with the propositional connectives, we use infix notation for $\dot\wedge$ and $\dot\vee$. Unlike with the propositional connectives, we assume the application of modal operators to have a higher precedence than regular functional application. So, for instance, we write $\dot\neg\langle r\rangle_2 \dot{y} \mathbin{\dot\vee} p \ x$ for $((\dot\neg(\langle r\rangle_2(\dot{y}))) \mathbin{\dot\vee} p)x$.

An *interpretation* is a function $\mathfrak{M}$ mapping B to the set $\{0,1\}$, S to a non-empty set, a functional type $\sigma\tau$ to the set of all total functions from the interpretation of σ to the interpretation of τ, interpreting all variables as elements of their respective types, and giving $\bot$, $\top$, $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\exists$, $\forall$, $\doteq$ their usual meaning. A *modal interpretation* is an interpretation that, in addition, satisfies the equations defining the constants $\sqsubseteq$, R, T, $\dot\neg$, $\dot\wedge$, $\dot\vee$, $\langle\_\rangle_n$, $[\_]_n$, E, A, $\dot{\_}$. If $\mathfrak{M}t = 1$, we say that $\mathfrak{M}$ *satisfies* t. A formula is called *satisfiable* if it has a satisfying modal interpretation.

3 Graded Hybrid Logic with Global Modalities

We begin with a tableau calculus for the restricted language without inclusion, reflexivity or transitivity assertions.

3.1 Tableaux and Evidence

For the sake of simplicity, we define our tableau calculus on negation normal expressions, i.e., terms of the form:

$$t \ ::= \ p \mid \dot\neg p \mid \dot{x} \mid \dot\neg\dot{x} \mid t \mathbin{\dot\wedge} t \mid t \mathbin{\dot\vee} t \mid \langle r\rangle_n t \mid [r]_n t \mid E_n t \mid A_n t$$

A *branch* Γ is a finite set of formulas s of the form

$$s \ ::= \ tx \mid rxy \mid DX \mid \bar{D}X \mid \bot$$

where t is a negation-normal modal expression of the above form. Formulas of the form rxy are called *accessibility formulas* or *edges*. We use the formula $\bot$ to explicitly mark unsatisfiable branches. We call a branch Γ *closed* if $\bot \in \Gamma$. Otherwise, Γ is called *open*. The branch consisting of the initial formula to be tested for satisfiability is called the *initial branch*.

Let Γ be a branch. With $\sim_\Gamma$ we denote the least equivalence relation $\sim$ on nominals such that $x \sim y$ for every formula $\bar{D}\{x,y\} \in \Gamma$. We define the *equational closure* $\tilde{\Gamma}$ of a branch Γ as

$$\begin{aligned}\tilde{\Gamma} \;:=\; & \Gamma \cup \{tx \mid \exists x' :\; x' \sim_\Gamma x \wedge tx' \in \Gamma\} \\ & \cup \{rxy \mid \exists x',\, y' :\; x' \sim_\Gamma x \wedge y' \sim_\Gamma y \wedge rx'y' \in \Gamma\}\end{aligned}$$

Clearly, $\tilde{\Gamma}$ is finite if Γ is finite. Reasoning with respect to $\tilde{\Gamma}$ can be implemented efficiently using disjoint-set forests, as demonstrated in [16].

A branch Γ is called *evident* if it satisfies all of the following *evidence conditions*:

$$\begin{aligned}
(t_1 \dot{\wedge} t_2)x \in \Gamma \;&\Rightarrow\; t_1x \in \tilde{\Gamma} \;\wedge\; t_2x \in \tilde{\Gamma} \\
(t_1 \dot{\vee} t_2)x \in \Gamma \;&\Rightarrow\; t_1x \in \tilde{\Gamma} \;\vee\; t_2x \in \tilde{\Gamma} \\
\langle r\rangle_n tx \in \Gamma \;&\Rightarrow\; \exists Y\!:\; |Y| = n+1 \;\wedge\; DY \in \Gamma \;\wedge\; \{rxy, ty \mid y \in Y\} \subseteq \tilde{\Gamma} \\
[r]_n tx \in \Gamma \;&\Rightarrow\; |\{y \mid rxy \in \tilde{\Gamma},\; ty \notin \tilde{\Gamma}\}/_{\sim_\Gamma}| \leq n \\
E_n tx \in \Gamma \;&\Rightarrow\; \exists Y\!:\; |Y| = n+1 \;\wedge\; DY \in \Gamma \;\wedge\; \{ty \mid y \in Y\} \subseteq \tilde{\Gamma} \\
A_n tx \in \Gamma \;&\Rightarrow\; |\{y \mid ty \notin \tilde{\Gamma}\}/_{\sim_\Gamma}| \leq n \\
\dot{x}y \in \Gamma \;&\Rightarrow\; x \sim_\Gamma y \\
\dot{\neg}\dot{x}y \in \Gamma \;&\Rightarrow\; x \not\sim_\Gamma y \\
\neg px \in \Gamma \;&\Rightarrow\; px \notin \tilde{\Gamma} \\
\bar{D}X \in \Gamma \;&\Rightarrow\; |X/_{\sim_\Gamma}| < |X| \\
DX \in \Gamma \;&\Rightarrow\; |X/_{\sim_\Gamma}| = |X|
\end{aligned}$$

Note that the evidence condition for $\bar{D}X \in \Gamma$ implies $|X| \geq 2$. A formula s is called *evident on* Γ if Γ satisfies the right-hand side of the evidence condition corresponding to s. For instance, $(t_1 \dot{\wedge} t_2)x$ is evident on Γ if and only if $\{t_1x, t_2x\} \subseteq \tilde{\Gamma}$.

We will now show that evident branches are satisfiable. Given a term t, we write $\mathcal{N}t$ for the set of nominals that occur in t. The notation is extended to sets of terms in the natural way: $\mathcal{N}\Gamma := \bigcup\{\mathcal{N}t \mid t \in \Gamma\}$.

Given a branch Γ, we construct the interpretation $\mathfrak{M}^\Gamma$ by taking as the domain of S the nominals on Γ, and interpreting propositional variables and roles as the smallest sets that are consistent with the respective assertions on Γ. To satisfy the equality constraints on Γ, all nominals that are equivalent modulo $\sim_\Gamma$ are mapped to the same fixed representative.

Let Γ be a branch and let $x_0 \in \mathcal{N}\Gamma$. Let ρ be a function from finite sets of nominals to nominals such that $\rho X \in X$ whenever X is nonempty. We define the interpretation $\mathfrak{M}^\Gamma$ as follows:

$$\begin{aligned}
&\mathfrak{M}^\Gamma \mathrm{S} := \mathcal{N}\Gamma \\
&\mathfrak{M}^\Gamma x := \text{if } x \in \mathcal{N}\Gamma \text{ then } \rho\{y \in \mathcal{N}\Gamma \mid y \sim_\Gamma x\} \text{ else } x_0 \\
&\mathfrak{M}^\Gamma p := \{x \in \mathcal{N}\Gamma \mid px \in \tilde{\Gamma}\} \\
&\mathfrak{M}^\Gamma r := \{(x,y) \in (\mathcal{N}\Gamma)^2 \mid rxy \in \tilde{\Gamma}\}
\end{aligned}$$

Note that in the last two lines of the definition, we interpret the set notation as a convenient description for the respective characteristic functions.

The evidence of $\langle r\rangle_n tx$ (and $E_n tx$) depends on the presence of structurally unrelated and possibly larger formulas DY ($|Y| = n{+}1$). Similar phenomena will be observed later with our tableau rules (see Fig. 1). Therefore, in the following we will need a measure $\lceil\_\rceil$ on formulas such that, in particular, $\lceil DY\rceil < \lceil\langle r\rangle_n tx\rceil$. Let $\lfloor s\rfloor$ denote the size of a formula s. We define the *order of* s, $\lceil s\rceil$, as follows:

$$\begin{array}{rcll} \lceil DX\rceil & := & 1 & \\ \lceil \bar{D}X\rceil & := & 1 & \text{if } |X| \leq 2 \\ \lceil \bar{D}X\rceil & := & 2 & \text{if } |X| > 2 \\ \lceil s\rceil & := & 3 + \lfloor s\rfloor & \text{otherwise} \end{array}$$

The case distinction in the definition of $\lceil\bar{D}X\rceil$ is exploited in the proofs of Theorems 3.3 and 4.4.

Theorem 3.1 (Model Existence). *If Γ is evident, then $\mathfrak{M}^\Gamma$ satisfies Γ.*

Proof. For every $s \in \Gamma$, we show that $\mathfrak{M}^\Gamma$ satisfies s by induction on the order of s. □

3.2 Tableau Rules

The tableau rules of our basic calculus $\mathcal{T}$ are defined in Fig. 1. In the rules, we write $\exists x \in X : \Gamma(x)$ for $\Gamma(x_1) \mid \ldots \mid \Gamma(x_n)$, where $X = \{x_1, \ldots, x_n\}$ and $\Gamma(x)$ is a set of formulas parameterized by x. In case $X = \emptyset$, the notation translates to $\perp$. Dually, we write $\forall x \in X : \Gamma(x)$ for $\Gamma(x_1), \ldots, \Gamma(x_n)$ ($X = \{x_1, \ldots, x_n\}$).

The side condition of $\mathcal{R}_\Diamond$ uses the notion of quasi-evidence, which we will introduce in Sect. 3.3. For now, we assume the rule is formulated with the restriction "$\langle r\rangle_n tx$ not evident on Γ".

Note that for $|X| < 2$ the rule $\mathcal{R}_{\bar{D}}$ instantiates to

$$\frac{\bar{D}X}{\perp}$$

A branch Γ is called a *proper extension* of a branch Δ if $\tilde{\Gamma} \supsetneq \tilde{\Delta}$. Note that if Γ is a proper extension of Δ, in particular it holds $\Gamma \supsetneq \Delta$. We implicitly restrict the applicability of the tableau rules such that a rule $\mathcal{R}$ is only applicable to a formula $s \in \Gamma$ if all of the alternative branches $\Delta_1, \ldots, \Delta_n$ resulting from this application are proper extensions of Γ.

Proposition 3.1 (Soundness). *Let $\Delta_1, \ldots, \Delta_n$ be the branches obtained from a branch Γ by a rule of $\mathcal{T}$. Then Γ is satisfiable if and only if there is some $i \in \{1, \ldots, n\}$ such that Δ_i is satisfiable.*

$$\mathcal{R}_{\dot\wedge}\ \frac{(s \mathbin{\dot\wedge} t)x}{sx,\ tx} \qquad\qquad \mathcal{R}_{\dot\vee}\ \frac{(s \mathbin{\dot\vee} t)x}{sx\ \mid\ tx}$$

$$\mathcal{R}_{\Diamond}\ \frac{\langle r\rangle_n tx}{DY,\ \forall y \in Y:\ rxy,\ ty}\ Y \text{ fresh},\ |Y| = n+1,\ \langle r\rangle_n tx \text{ not quasi-evident on } \mathit{\Gamma}$$

$$\mathcal{R}_{\Box}\ \frac{[r]_n tx}{\bar{D}Y\ \mid\ \exists y \in Y:\ ty}\ Y \subseteq \{y \mid rxy \in \tilde{\mathit{\Gamma}}\},\ |Y| = |Y/{\sim_{\mathit{\Gamma}}}| = n+1$$

$$\mathcal{R}_E\ \frac{E_n tx}{DY,\ \forall y \in Y:\ ty}\ Y \text{ fresh},\ |Y| = n+1,\ E_n tx \text{ not evident on } \mathit{\Gamma}$$

$$\mathcal{R}_A\ \frac{A_n tx}{\bar{D}Y\ \mid\ \exists y \in Y:\ ty}\ Y \subseteq \mathcal{N}\mathit{\Gamma},\ |Y| = |Y/{\sim_{\mathit{\Gamma}}}| = n+1 \qquad \mathcal{R}_N\ \frac{\dot{x}y}{\bar{D}\{x,y\}}\ x \neq y$$

$$\mathcal{R}_{\bar{N}}\ \frac{\dot\neg\dot{x}y}{D\{x,y\}}\ x \neq y \qquad \mathcal{R}_{\bar{D}}\ \frac{\bar{D}X}{\exists x, y \in X,\ x \neq y:\ \bar{D}\{x,y\}} \qquad \mathcal{R}_D\ \frac{DX}{\perp}\ |X/{\sim_{\mathit{\Gamma}}}| < |X|$$

$$\mathcal{R}^{\perp}_{\dot\neg}\ \frac{\dot\neg px}{\perp}\ px \in \tilde{\mathit{\Gamma}} \qquad\qquad \mathcal{R}^{\perp}_{\bar{N}}\ \frac{\dot\neg\dot{x}x}{\perp}$$

$\mathit{\Gamma}$ is the branch to which a rule is applied. "Y fresh" stands for $Y \cap \mathcal{N}\mathit{\Gamma} = \emptyset$.

Fig. 1. Tableau rules for $\mathcal{T}$

3.3 Control

The restrictions on the applicability of the tableau rules given by the evidence conditions are not sufficient for termination. To obtain a terminating calculus, the rule $\mathcal{R}_{\Diamond}$ needs to be restrained further. We do so by weakening the notion of evidence for diamond formulas. The weaker notion, called quasi-evidence, is then used in the side condition of $\mathcal{R}_{\Diamond}$ in place of evidence. Quasi-evidence must be weak enough to guarantee termination of the calculus but strong enough to preserve completeness.

The notions of quasi-evidence used in previous work on pattern-based blocking [15,14] turn out to be too weak in the presence of graded modalities. For instance, intuitively adapting the notion in [15] would give us the following candidate definition:

A formula $\langle r\rangle_m sx$ is quasi-evident on $\mathit{\Gamma}$ if there are $y, z_1, \ldots, z_m$ such that $\{ryz_1, sz_1, \ldots, ryz_m, sz_m\} \subseteq \tilde{\mathit{\Gamma}}$ and $\{[r]_n ty \mid [r]_n tx \in \tilde{\mathit{\Gamma}}\} \subseteq \tilde{\mathit{\Gamma}}$. (We also say: $\langle r\rangle_m sx$ is quasi-evident if the corresponding *pattern* $\{\langle r\rangle_m s\} \cup \{[r]_n t \mid [r]_n tx \in \tilde{\mathit{\Gamma}}\}$ is *expanded*).

With this definition of quasi-evidence, no rule of our calculus would apply to the following branch:

$$\mathit{\Gamma} := \{ryz,\ qz,\ [r]_1(p \mathbin{\dot\wedge} \dot\neg p)y,\ \langle r\rangle_0 qx,\ [r]_1(p \mathbin{\dot\wedge} \dot\neg p)x,\ rxu,\ \dot\neg qu\}$$

As Γ is clearly unsatisfiable, the notion of quasi-evidence needs to be adapted.

Given a branch Γ and a role r, an *r-pattern* is a set of expressions of the form μs, where $\mu \in \{\langle r\rangle_n, [r]_n \mid n \in \mathbb{N}\}$. We write $P^r_\Gamma x$ for the largest r-pattern P such that $P \subseteq \{t \mid tx \in \tilde{\Gamma}\}$. We call $P^r_\Gamma x$ the r-pattern of x on Γ. An r-pattern P is *expanded on Γ* if there are nominals x, y such that $rxy \in \tilde{\Gamma}$ and $P \subseteq P^r_\Gamma x$. In this case, we say that the nominal x *expands P on Γ*.

A diamond formula $\langle r\rangle_n sx \in \Gamma$ is *quasi-evident on Γ* if it is either evident on Γ or *x has no r-successor on Γ* (i.e., there is no y such that $rxy \in \tilde{\Gamma}$) and $P^r_\Gamma x$ is expanded on Γ. The rule $\mathcal{R}_\Diamond$ can only be applied to diamond formulas that are not quasi-evident.

Note that whenever $\langle r\rangle_n sx \in \Gamma$ is quasi-evident but not evident on Γ, there is a nominal y that expands $P^r_\Gamma x$ on Γ.

We call a branch Γ quasi-evident if it satisfies all of the evidence conditions but the one for diamond formulas, which we replace by:

$$\langle r\rangle_n tx \in \Gamma \;\Rightarrow\; \langle r\rangle_n tx \text{ is quasi-evident on } \Gamma$$

One can show the following lemma:

Lemma 3.1. *Let Γ be a quasi-evident branch and let $\langle r\rangle_n sx \in \Gamma$ be not evident on Γ. Let y be a nominal that expands $P^r_\Gamma x$ on Γ and $\Delta := \Gamma \cup \{rxz \mid ryz \in \tilde{\Gamma}\}$. Then:*

1. $\forall z :\; rxz \in \tilde{\Delta} \iff ryz \in \tilde{\Gamma}$,
2. $\forall m, t :\; \langle r\rangle_m t \in P^r_\Gamma x \implies \langle r\rangle_m tx$ *evident on* Δ,
3. $\langle r\rangle_n sx$ *evident on* Δ,
4. $\forall r', m, t, z :\; \langle r'\rangle_m tz$ *evident on* $\Gamma \implies \langle r'\rangle_m tz$ *evident on* Δ,
5. Δ *quasi-evident.*

Theorem 3.2 (Evidence Completion). *For every quasi-evident branch Γ there is an evident branch Δ such that $\Gamma \subseteq \Delta$.*

Proof. For every branch Γ, we define:

$$\varphi\Gamma := |\{\langle r\rangle_n sx \mid \langle r\rangle_n sx \in \Gamma \wedge \langle t\rangle_n sx \text{ not evident on } \Gamma\}|$$

Let Γ be quasi-evident. We proceed by induction on $\varphi\Gamma$. If $\varphi\Gamma = 0$, then Γ is evident and we are done. Otherwise, there is a diamond $\langle r\rangle_n sx \in \Gamma$ that is not evident on Γ. Let y be a nominal that expands $P^r_\Gamma x$ on Γ, and let $\Gamma' := \Gamma \cup \{rxz \mid ryz \in \tilde{\Gamma}\}$. By Lemma 3.1(3-5), Γ' is quasi-evident and $\varphi\Gamma' < \varphi\Gamma$. So, by the inductive hypothesis, there is some evident branch Δ such that $\Delta \supseteq \Gamma' \supseteq \Gamma$. □

A branch is called *maximal* if it cannot be extended by any tableau rule.

Theorem 3.3 (Quasi-evidence). *Every open and maximal branch in $\mathcal{T}$ is quasi-evident.*

Proof. Let Γ be an open and maximal branch. We show that every $s \in \Gamma$ that is not of the form px or rxy is (quasi-)evident on Γ by induction on the order of s. □

3.4 Termination

We will now show that every tableau derivation is finite. As usual, the main difficulty is bounding the number of applications of generative rules, in particular of $\mathcal{R}_{\Diamond}$. The present proof is notably more complex than the proofs in [15,14] since now, an application of $\mathcal{R}_{\Diamond}$ does not necessarily expand a new pattern. Hence, we need to combine the pattern-counting argument from [15,14] with a bound on the number of non-expanding applications of $\mathcal{R}_{\Diamond}$.

Since the rules $\mathcal{R}_{\dot{\vee}}$, $\mathcal{R}_{\Box}$, $\mathcal{R}_{A}$, and $\mathcal{R}_{\bar{D}}$ are all finitely branching, by König's lemma it suffices to show that the construction of every individual branch terminates. Since tableau rule application always produces proper extensions of branches, it then suffices to show that the size (i.e., cardinality) of an individual branch is bounded.

First, we show that the size of a branch Γ is bounded by a function in the number of nominals on Γ. Then, we show that this number itself is bounded from above, completing the termination proof.

We write $\Gamma \xrightarrow{\mathcal{R}} \Delta$ to denote that the branch Δ is obtained from Γ by the rule $\mathcal{R}$. We write $\Gamma \to \Delta$ if Δ is obtained from Γ by a single rule application. We write $\mathcal{S}\Gamma$ for the set of all modal expressions occurring on Γ, possibly as subterms of other expressions, and $\mathrm{Rel}\,\Gamma$ for the set of all roles that occur on Γ.

Crucial for the termination argument is the fact the the tableau rules cannot introduce any modal expressions that do not already occur on the initial branch.

Proposition 3.2. *If Γ, Δ are branches such that Δ is obtained from Γ by any rule of $\mathcal{T}$, then $\mathcal{S}\Delta = \mathcal{S}\Gamma$.*

Let $m_0 = \max\{n \mid \exists r, s, x :\ \langle r \rangle_n sx \in \Gamma \vee [r]_n sx \in \Gamma\}$. For every pair of nominals x, y and every role r, a branch Γ may contain an edge rxy, for every set $X \subseteq \mathcal{N}\Gamma$ where $|X| \le m_0$, Γ may contain constants DX and $\bar{D}X$ and, for every expression $s \in \mathcal{S}\Gamma$, a formula sx. Hence, the size of Γ is bounded by $|\mathrm{Rel}\,\Gamma| \cdot |\mathcal{N}\Gamma|^2 + 2m_0 \cdot |\mathcal{N}\Gamma|^{m_0} + |\mathcal{S}\Gamma| \cdot |\mathcal{N}\Gamma|$. By Proposition 3.2, we know that $|\mathcal{S}\Gamma|$ and $|\mathrm{Rel}\,\Gamma|$ depend only on the initial branch.

Note that the above bound is exponential in m_0. If, however, we represented distinctness constraints by binary equations and disequations, we could easily give a bound that is independent from m_0 by replacing the summand $2m_0 \cdot |\mathcal{N}\Gamma|^{m_0}$ with $2|\mathcal{N}\Gamma|^2$.

By the above, it suffices to show that $|\mathcal{N}\Gamma|$ is exponentially bounded in the size of the initial formula. We do so by giving a bound on the number of applications of $\mathcal{R}_{\Diamond}$ and $\mathcal{R}_E$ that can occur in the derivation of a branch, which suffices since $\mathcal{R}_{\Diamond}$ and $\mathcal{R}_E$ are the only two rules that can introduce new nominals.

We begin by showing that $\mathcal{R}_E$ can be applied at most as many times, as there are distinct modal expressions of the form $E_n s$ on the initial branch. For this purpose, we define a function ψ_E such that $\psi_E\Gamma := \{E_n s \in \mathcal{S}\Gamma \mid \exists x \in \mathcal{N}\Gamma :\ E_n sx \text{ not evident on } \Gamma\}$. Since $|\psi_E\Gamma|$ is bounded from below by 0, it suffices to show that the number decreases with every application of $\mathcal{R}_E$ (and is non-increasing otherwise, which is obvious).

Proposition 3.3. $\Gamma \overset{\mathcal{R}_E}{\to} \Delta \implies |\psi_E \Gamma| > |\psi_E \Delta|$

The proof proceeds analogously to the corresponding arguments in [15,14].

Now we show that $\mathcal{R}_\Diamond$ can be applied at most finitely often in a derivation. Since there are only finitely many roles, it suffices to show that $\mathcal{R}_\Diamond$ can be applied at most finitely often for each role. Observe that since $\mathcal{R}_\Diamond$ is only applicable to diamond formulas that are not quasi-evident, it holds:

Proposition 3.4. *If $\mathcal{R}_\Diamond$ is applicable to a formula $\langle r \rangle_n sx \in \Gamma$, then either*

1. *x has an r-successor on Γ, or*
2. *$P^r_\Gamma x$ is not expanded on Γ.*

Let Γ and Δ be branches such that Δ is obtained from Γ by applying $\mathcal{R}_\Diamond$ to a formula $\langle r \rangle_n sx \in \Gamma$ such that $P^r_\Gamma x$ is not expanded on Γ. It is easy to see that $P^r_\Delta x$ must be expanded on Δ. Let us call such an application of $\mathcal{R}_\Diamond$ *pattern-expanding.*

Let $\mathrm{Pat}^r\Gamma := \mathcal{P}(\{\langle r \rangle_n s \in \mathcal{S}\Gamma\} \cup \{[r]_n s \in \mathcal{S}\Gamma\})$. In other words, $\mathrm{Pat}^r\Gamma$ contains all the possible sets of r-diamonds and r-boxes from $\mathcal{S}\Gamma$. Since $\Gamma \to \Delta$ implies $\tilde{\Gamma} \subseteq \tilde{\Delta}$, it holds:

Lemma 3.2. *Let $\Gamma \to \Delta$ and $P \in \mathrm{Pat}^r\Gamma$. If P is expanded on Γ, then P is expanded on Δ.*

So, for each role r the derivation of a branch has at most $|\mathrm{Pat}^r\Gamma_0|$ pattern-expanding applications of $\mathcal{R}_\Diamond$, where Γ_0 is the initial branch. Clearly, $|\mathrm{Pat}^r\Gamma_0|$ is exponentially bounded in the size of the initial formula.

Hence, it remains to show that a derivation can contain only finitely many applications of $\mathcal{R}_\Diamond$ assuming that none of the applications is pattern-expanding. We say a nominal x has a *successor* on Γ if x has an r-successor on Γ for any role r. A set of nominals X has a successor on Γ if there is some $x \in X$ that has a successor on Γ. We define

$$\psi^X_\Diamond \Gamma := |\{\langle r \rangle_n s \in \mathcal{S}\Gamma \mid \exists x \in X : \ \langle r \rangle_n sx \text{ not evident on } \Gamma\}|$$

and

$$\psi_\Diamond \Gamma := \sum_{\substack{X \in \mathcal{N}\Gamma/\sim_\Gamma \\ X \text{ has a successor on } \Gamma}} \psi^X_\Diamond \Gamma \ .$$

Proposition 3.5. *Let $\Gamma \to \Delta$ such that Δ is obtained from Γ by some rule application other than a pattern-expanding application of $\mathcal{R}_\Diamond$.*

1. *If Δ is obtained from Γ by $\mathcal{R}_\Diamond$, then $\psi_\Diamond \Gamma > \psi_\Diamond \Delta$.*
2. *Otherwise, $\psi_\Diamond \Gamma \geq \psi_\Diamond \Delta$.*

This completes the termination proof. Since the cardinalities of the sets $\mathrm{Pat}^r\Gamma$ are exponentially bounded in the size n_0 of the initial formula, $|\psi_E \Gamma|$ is polynomial in n_0, and $\psi_\Diamond \Gamma$ polynomial in $|\Gamma|$ and n_0, $|\mathcal{N}\Gamma|$ is exponentially bounded in n_0. Since $|\Gamma|$ is polynomial in $|\mathcal{N}\Gamma|$, we conclude that $|\Gamma|$ is at most exponential in n_0. By cumulativity, the construction of Γ terminates in at most exponentially many steps in n_0. This suffices to give us a NExpTime complexity bound for the decision procedure based on the calculus.

4 Adding Reflexivity, Transitivity and Role Inclusion

We now extend $\mathcal{T}$ to deal with reflexivity, transitivity and inclusion assertions. As in related work on description logic [5,8,10,9], we restrict our modal expressions to contain no graded boxes for roles that have transitive subroles.

We define $\subseteq^*_\Gamma$ as the smallest reflexive and transitive relation such that $r \subseteq^*_\Gamma r'$ whenever $r \sqsubseteq r' \in \Gamma$. A role r is called *simple* if there is no r' such that $r' \subseteq^*_\Gamma r$ and $Tr' \in \Gamma$. Observe that all subroles of a simple role are in turn simple.

Our branches may now contain inclusion, reflexivity and transitivity assertions:

$$s \ ::= \ tx \mid rxy \mid DX \mid \bar{D}X \mid \bot \mid r \sqsubseteq r' \mid Rr \mid Tr$$

The modal expressions t in formulas of the form tx are restricted to contain no boxes $[r]_n s$ with $n > 0$ unless r is simple.

Following the ideas in [5,8,10,9], we introduce the *induced transition relation* $\trianglerighteq^r_\Gamma$ to reason about accessibility in the presence of inclusion axioms. Intuitively, $x \trianglerighteq^r_\Gamma y$ means that in every model of Γ, y is accessible from x via r.

4.1 Extending Evidence

To account for the new types of formulas, we extend the evidence conditions as follows:

$$\begin{aligned} r \sqsubseteq r' \in \Gamma \ &\Rightarrow \ \forall x, y \in \mathcal{N}\Gamma : \ rxy \in \tilde{\Gamma} \Rightarrow r'xy \in \tilde{\Gamma} \\ Rr \in \Gamma \ &\Rightarrow \ \forall x \in \mathcal{N}\Gamma : \ rxx \in \tilde{\Gamma} \\ Tr \in \Gamma \ &\Rightarrow \ \forall x, y, z \in \mathcal{N}\Gamma : \ rxy \in \tilde{\Gamma} \wedge ryz \in \tilde{\Gamma} \Rightarrow rxz \in \tilde{\Gamma} \end{aligned}$$

It is easy to see that if Γ satisfies the extended evidence conditions, $\mathfrak{M}^\Gamma$ will satisfy the new formulas. Hence, Theorem 3.1 adapts to the extended system.

Theorem 4.1 (Model Existence). *If Γ is evident, then $\mathfrak{M}^\Gamma$ satisfies Γ.*

4.2 Pre-evidence

To account for the new evidence conditions, one could imagine the following rules.

$$\frac{r \sqsubseteq r', \ rxy}{r'xy} \qquad\qquad \frac{Rr}{rxx}\ x \in \mathcal{N}\Gamma \qquad\qquad \frac{Tr, \ rxy, \ ryz}{rxz}$$

In the presence of blocking, however, the rules are problematic. In particular, the rule for reflexivity renders the notion of quasi-evidence that we use for $\mathcal{T}$ ineffective to ensure termination. Once we add a reflexive edge rxx to a branch Γ, x will have an r-successor on Γ, meaning quasi-evidence will coincide with evidence for all r-diamonds on x. Similarly, the rule for transitivity is known to be incomplete in the presence of blocking [14].

We solve the problem by defining a weaker notion of evidence, called *pre-evidence*. To satisfy the pre-evidence conditions, we do not have to explicitly add reflexive or transitive edges during tableau construction. We will extend our tableau rules and the notion of quasi-evidence such that every open and maximal branch in the extended calculus can be completed to a pre-evident branch, which in turn can be made evident by adding the implicit edges.

We define the relation $\rhd^r_\varGamma$ as the least relation such that:

$$\begin{aligned} rxy \in \tilde{\varGamma} \quad &\Rightarrow \quad x \rhd^r_\varGamma y \\ r' \sqsubseteq r \in \varGamma,\ x \rhd^{r'}_\varGamma y \quad &\Rightarrow \quad x \rhd^r_\varGamma y \end{aligned}$$

The relation $\rhd^r_\varGamma$ does not account for reflexivity. To do so, we extend it as follows:

$$\trianglerighteq^r_\varGamma \;:=\; \begin{cases} \rhd^r_\varGamma \cup \{(x,y) \,|\, x,y \in \mathcal{N}\varGamma \wedge x \sim_\varGamma y\} & \text{if } \exists r' :\ r' \subseteq^*_\varGamma r \wedge Rr' \in \varGamma \\ \rhd^r_\varGamma & \text{otherwise} \end{cases}$$

The *pre-evidence conditions* are obtained from the evidence conditions by omitting the conditions for inclusion and reflexivity assertions and replacing the conditions for diamonds, boxes and transitivity assertions as follows:

$$\begin{aligned} \langle r\rangle_n tx \in \varGamma \;&\Rightarrow\; \exists Y\colon\ |Y| = n+1 \ \wedge\ DY \in \tilde{\varGamma} \ \wedge\ \forall y \in Y :\ x \trianglerighteq^r_\varGamma y \wedge ty \in \tilde{\varGamma} \\ [r]_n tx \in \varGamma \;&\Rightarrow\; |\{y \,|\, x \trianglerighteq^r_\varGamma y,\ ty \notin \tilde{\varGamma}\}/_{\sim_\varGamma}| \le n \\ Tr \in \varGamma \;&\Rightarrow\; \forall x,y :\ [r']_0 tx \in \tilde{\varGamma} \wedge r \subseteq^*_\varGamma r' \wedge x \rhd^r_\varGamma y \Rightarrow [r]_0 ty \in \tilde{\varGamma} \end{aligned}$$

Note that we do not need pre-evidence conditions for inclusion or reflexivity assertions as their semantics is taken care of by the way we define the relation $x \trianglerighteq^r_\varGamma y$. Pre-evidence of individual formulas is defined analogously to the corresponding evidence notion.

We now show that every pre-evident branch can be extended to an evident branch. Let the *evidence closure* $\hat{\varGamma}$ of a branch $\varGamma$ be defined as the least superset of $\varGamma$ such that:

$$\begin{aligned} x \trianglerighteq^r_\varGamma y \;&\Rightarrow\; rxy \in \hat{\varGamma} \\ Tr \in \hat{\varGamma} \wedge rxy \in \hat{\varGamma} \wedge ryz \in \hat{\varGamma} \;&\Rightarrow\; rxz \in \hat{\varGamma} \\ r \sqsubseteq r' \in \hat{\varGamma} \wedge rxy \in \hat{\varGamma} \;&\Rightarrow\; r'xy \in \hat{\varGamma} \end{aligned}$$

Note that by construction, we have $rxy \in \hat{\hat{\varGamma}} \iff rxy \in \hat{\varGamma}$.

Lemma 4.1. *Let $\varGamma$ be a branch and r be simple on $\varGamma$. Then $x \trianglerighteq^r_\varGamma y \iff rxy \in \hat{\varGamma}$*

Lemma 4.2. *Let $\varGamma$ be a branch and let $rxy \in \hat{\varGamma}$. Then either $x \trianglerighteq^r_\varGamma y$, or there is an r' such that $\{r' \sqsubseteq r, Tr'\} \subseteq \varGamma$ and*

$$\exists\, n{\ge}2\ \exists x_1, \dots, x_n\colon\ x_1 = x \ \wedge\ x_n = y \ \wedge\ \forall 1{\le}i{<}n :\ x_i \rhd^{r'}_\varGamma x_{i+1} \ .$$

Theorem 4.2 (Evidence Completion). *$\varGamma$ pre-evident $\Longrightarrow$ $\hat{\varGamma}$ evident*

Proof. Straightforward, using Lemmas 4.1 and 4.2. □

$$\mathcal{R}_\Box \ \frac{[r]_n tx}{\bar{D}Y \ \mid \ \exists y \in Y : \ ty} \ Y \subseteq \{y \,|\, x \trianglerighteq^r_\Gamma y\},\ |Y| = |Y/_{\sim_\Gamma}| = n+1$$

$$\mathcal{R}_T \ \frac{Tr,\ [r']_0 tx}{[r]_0 ty} \ r \subseteq^*_\Gamma r',\ x \triangleright^r_\Gamma y$$

Fig. 2. New rules for $\mathcal{T}_\sqsubseteq$

4.3 Tableau Rules

The tableau rules for the extended calculus $\mathcal{T}_\sqsubseteq$ in Fig. 2 replace the original rule $\mathcal{R}_\Box$ from Fig. 1 and add a new rule $\mathcal{R}_T$, which is necessary to achieve the pre-evidence condition for transitivity assertions. While the formulation of $\mathcal{R}_\Diamond$ remains unchanged, the rule will now have to use an adapted notion of quasi-evidence, which will be introduced in Sect. 4.4. For now, we assume $\mathcal{R}_\Diamond$ is formulated with the restriction "$\langle r\rangle_n tx$ not pre-evident on Γ" instead. Again, it is not hard to verify that the extended rules are sound.

4.4 Control

As it turns out, in the presence of role inclusion we have to modify the definition of patterns. It no longer suffices to consider patterns separately for each role. This is due to the fact that now, different roles may be constrained by inclusion assertions. Consider, for instance, the unsatisfiable branch

$$\Gamma := \{r \sqsubseteq r',\ \langle r\rangle_0 px,\ \langle r'\rangle_0 \dot{\neg} px,\ [r']_1 (p \mathbin{\dot\wedge} \dot{\neg} p)x,\ r'xy,\ \dot{\neg} py,\ \langle r\rangle_0 pz,\ rzu,\ pu\}$$

According to our previous notion of quasi-evidence, $\langle r\rangle_0 px$ is quasi-evident on Γ as x has no r-successor (even if we extend the set of successors to $\{y \,|\, x \triangleright^r_\Gamma y\}$) and $P^r_\Gamma x$ is expanded. Since the other two diamonds on Γ are evident, Γ is quasi-evident, witnessing the incompleteness of our previous definition of patterns.

Hence, we redefine the notion of a pattern as follows. Given a branch Γ, a *pattern* is a set of terms of the form μs, where $\mu \in \{\langle r\rangle_n, [r]_n \,|\, r \in \mathrm{Rel}\,\Gamma,\ n \in \mathbb{N}\}$. We write $P_\Gamma x$ for the largest pattern P such that $P \subseteq \{t \,|\, tx \in \tilde{\Gamma}\}$. We call $P_\Gamma x$ the pattern of x on Γ. A pattern P is *expanded on* Γ if there are nominals x, y and a role r such that $x \triangleright^r_\Gamma y$ and $P \subseteq P_\Gamma x$. In this case, we say that x *expands* P *on* Γ. Note that here we use the relation $\triangleright^r_\Gamma$ rather than $\trianglerighteq^r_\Gamma$. Otherwise, we would get the same problems with termination as outlined in Sect. 4.2.

A diamond formula $\langle r\rangle_n sx$ is *quasi-evident on* Γ if it is either pre-evident on Γ or x has no *successor* on Γ (i.e., there is no y such that for any r, $x \triangleright^r_\Gamma y$) and $P_\Gamma x$ is expanded on Γ. As before, we restrain the rule $\mathcal{R}_\Diamond$ such that it can only be applied to diamond formulas that are not quasi-evident, and call a branch Γ quasi-evident if it satisfies all of the pre-evidence conditions but the one for diamond formulas, which we again replace by

$$\langle r\rangle_n tx \in \Gamma \ \Rightarrow \ \langle r\rangle_n tx \text{ is quasi-evident on } \Gamma$$

but now with the adapted notion of quasi-evidence.

Lemma 4.3. *Let x, y, u, v be nominals and Γ, Δ branches such that $\{r \mid rxy \in \tilde{\Gamma}\} = \{r \mid ruv \in \tilde{\Delta}\}$. Then, for every r, $x \rhd^r_\Gamma y \Leftrightarrow u \rhd^r_\Delta v$.*

Lemma 4.4. *Let Γ be a quasi-evident branch and let $\langle r \rangle_n sx$ be not pre-evident on Γ. Let y expand $P_\Gamma x$ on Γ and $\Delta := \Gamma \cup \{r'xz \mid r'yz \in \tilde{\Gamma}\}$. Then:*

1. *$\forall r', z :\ x \rhd^{r'}_\Delta z \iff y \rhd^{r'}_\Gamma z$ and $x \unrhd^{r'}_\Delta z \iff y \unrhd^{r'}_\Gamma z$,*
2. *$\forall r', m, t :\ \langle r' \rangle_m t \in P_\Gamma x \implies \langle r' \rangle_m tx$ pre-evident on Δ,*
3. *$\langle r \rangle_n sx$ pre-evident on Δ,*
4. *$\forall r', m, t, z :\ \langle r' \rangle_m tz$ pre-evident on $\Gamma \implies \langle r' \rangle_m tz$ pre-evident on Δ,*
5. *Δ quasi-evident.*

Proof. Analogous to the proof of Lemma 3.1, Lemma 4.3 being used for (1). □

Theorem 4.3 (Pre-evidence Completion). *For every quasi-evident branch Γ there is a pre-evident branch Δ such that $\Gamma \subseteq \Delta$.*

Proof. Proceeds analogously to the proof of Theorem 3.2 with Lemma 4.4 in place of Lemma 3.1. □

Theorem 4.4 (Quasi-evidence). *Every open and maximal branch in $\mathcal{T}_\sqsubseteq$ is quasi-evident.*

Proof. Proceeds analogously to the proof of Theorem 3.3. □

While requiring some adaptations, the termination proof for $\mathcal{T}_\sqsubseteq$ is mostly analogous to the proof for $\mathcal{T}$.

5 Conclusion

We have presented a terminating tableau calculus for graded hybrid logic with global modalities and role hierarchies. Following [19,20,14], our calculus is cumulative, representing state equality abstractly via an equivalence relation (declarative approach). The existing calculi for equivalent and stronger logics [8,10,9] work on possibly cyclic graph structures and treat equality by destructive graph transformation during tableau construction (procedural approach). The procedural approach encompasses algorithmic decisions that are not present in the more abstract declarative approach. From a declarative calculus we can always obtain a procedural system by refinement.

Exploiting an extended pattern-based blocking technique and the cumulativity of our calculus, we have proved a NExpTime complexity bound for the associated decision procedure. To ensure termination of pattern-based blocking in the presence of reflexivity, we differentiated between the induced transition relation $\unrhd^r_\Gamma$ and its non-reflexive counterpart $\rhd^r_\Gamma$. The implementation of pattern-based blocking for a hybrid language with global modalities [16] reveals its considerable practical potential. We consider it a promising project to implement the extended version of pattern-based blocking presented in this paper and compare its performance to that of established blocking techniques.

References

1. Fine, K.: In so many possible worlds. Notre Dame J. Form. Log. 13(4), 516–520 (1972)
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications, 2nd edn. Cambridge University Press, Cambridge (2007)
3. Areces, C., ten Cate, B.: Hybrid logics. In: [21], pp. 821–868
4. Goranko, V., Passy, S.: Using the universal modality: Gains and questions. J. Log. Comput. 2(1), 5–30 (1992)
5. Horrocks, I.: Optimising Tableaux Decision Procedures for Description Logics. PhD thesis, University of Manchester (1997)
6. Sattler, U.: A concept language extended with different kinds of transitive roles. In: Görz, G., Hölldobler, S. (eds.) KI 1996. LNCS (LNAI), vol. 1137. Springer, Heidelberg (1996)
7. Baader, F., Lutz, C.: Description logic. In: [21], pp. 757–820
8. Horrocks, I., Sattler, U.: Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In: Nebel, B. (ed.) Proc. 17th Intl. Joint Conf. on Artificial Intelligence (IJCAI 2001), pp. 199–204. Morgan Kaufmann, San Francisco (2001)
9. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proc. 10th Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 57–67. AAAI Press, Menlo Park (2006)
10. Horrocks, I., Sattler, U.: A tableau decision procedure for $\mathcal{SHOIQ}$. J. Autom. Reasoning 39(3), 249–276 (2007)
11. Kripke, S.A.: Semantical analysis of modal logic I: Normal modal propositional calculi. Z. Math. Logik Grundlagen Math. 9, 67–96 (1963)
12. Baader, F., Buchheit, M., Hollunder, B.: Cardinality restrictions on concepts. Artif. Intell. 88(1–2), 195–213 (1996)
13. Motik, B., Shearer, R., Horrocks, I.: Optimized reasoning in description logics using hypertableaux. In: Pfenning, F. (ed.) CADE-21. LNCS (LNAI), vol. 4603, pp. 67–83. Springer, Heidelberg (2007)
14. Kaminski, M., Smolka, G.: Terminating tableau systems for hybrid logic with difference and converse. J. Log. Lang. Inf. (to appear, 2009)
15. Kaminski, M., Smolka, G.: Hybrid tableaux for the difference modality. In: Areces, C., Demri, S. (eds.) Proc. 5th Workshop on Methods for Modalities (M4M5 2007). Electr. Notes Theor. Comput. Sci., vol. 231, pp. 241–257. Elsevier, Amsterdam (2009)
16. Götzmann, D.: Spartacus: A Tableau Prover for Hybrid Logic. M.Sc. thesis, Saarland University (2009)
17. Kaminski, M., Smolka, G.: Terminating tableaux for hybrid logic with the difference modality and converse. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 210–225. Springer, Heidelberg (2008)
18. Farmer, W.M.: The seven virtues of simple type theory. J. Appl. Log. 6(3), 267–286 (2008)
19. Bolander, T., Braüner, T.: Tableau-based decision procedures for hybrid logic. J. Log. Comput. 16(6), 737–763 (2006)
20. Bolander, T., Blackburn, P.: Termination for hybrid tableaus. J. Log. Comput. 17(3), 517–554 (2007)
21. Blackburn, P., van Benthem, J., Wolter, F. (eds.): Handbook of Modal Logic. Studies in Logic and Practical Reasoning, vol. 3. Elsevier, Amsterdam (2007)

Prime Implicate Tries*

Andrew Matusiewicz[1], Neil V. Murray[1], and Erik Rosenthal[2]

[1] Institute for Informatics, Logics, & Security Studies, Department of Computer Science,
University at Albany – SUNY, Albany, NY 12222, USA
nvm@cs.albany.edu, a_matusi@cs.albany.edu

[2] Department of Mathematics, University of New Haven, West Haven, CT 06516, USA
erosenthal@newhaven.edu

Abstract. The *prime implicate trie* (pi-trie) of a logical formula is a tree whose branches are labeled with the prime implicates of the formula. The technology of reduced implicate tries is employed to analyze the structure of pi-tries. Appropriate lemmas and theorems are proved, and an algorithm that builds the pi-trie from a logical formula is developed. Preliminary experimental results are presented.

1 Introduction

The *prime implicate trie* (pi-trie) of a logical formula, developed in this paper, is a tree whose branches are labeled with the prime implicates of the formula. The *reduced implicate trie* (ri-trie) is a data structure that was introduced in [10] as a target language for knowledge compilation. It has the property that, even when large, a query can always be processed in time *linear in the size of the query*. The pi-trie provides compact storage for prime implicates, while the ri-trie stores information for answering arbitrary queries. Nonetheless, the two are structurally similar, and, in the sequel, techniques originally developed to produce ri-tries are modified to produce pi-tries. The resulting algorithm is quite different from any other prime implicate algorithm of which the authors are aware.

Consequences expressed as minimal clauses that are implied by a formula are its *prime implicates*, while minimal conjunctions of literals that imply a formula are its *prime implicants*. Implicates are useful in certain approaches to non-monotonic reasoning [7,14,16], where all consequences of a formula — for example, the support set for a proposed common-sense conclusion — are required. Another application is error analysis during hardware verification, where satisfying models are desired. Many algorithms have been developed to compute the prime implicates (or implicants) of a propositional boolean formula — see, for example, [1,2,3,4,5,6,8,13,15,17,18].

The properties of reduced implicate tries are reviewed in Section 2. In Section 3 prime implicate tries, which are subtries of ri-tries, are introduced. An algorithm that produces pi-tries is developed in Section 3.2.

* This research was supported in part by the National Science Foundation under grants IIS-0712849 and IIS-0712752.

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 250–264, 2009.

2 Reduced Implicate Tries

The terminology used in this paper for logical formulas is standard: An *atom* is a propositional variable, a *literal* is an atom or the negation of an atom, and a *clause* is a disjunction of literals.[1] Clauses are often referred to as sets of literals. Many authors restrict the theory to *conjunctive normal form* (CNF) — a conjunction of clauses — but no such restriction is required in this paper. An *implicate* of a logical formula is a clause entailed by the formula, and a non-tautological clause is a *prime implicate* if no proper subset is an implicate. Thus a clause C is an implicate of a logical formula $\mathcal{F}$ if and only if C is satisfied by every interpretation that satisfies $\mathcal{F}$. Hence, asking whether a given clause is entailed by a formula is equivalent to the question, *Is the clause an implicate of the formula*?

A tautology is logically equivalent to the empty sentence (empty conjunction) and thus has no implicates. A contradiction, on the other hand, is logically equivalent to the empty clause (empty disjunction). Thus all clauses are implicates, and the empty clause is the only prime implicate.

2.1 Reduced Implicate Tries

The trie is a well-known data structure introduced by Morrison in 1968 [9]; it is a tree in which each branch represents the sequence of symbols labeling the nodes[2] on that branch, in descending order. Tries have been used to represent logical formulas, including sets of prime implicates [16]. The nodes along each branch represent the literals of a clause, and the conjunction of all such clauses is a CNF equivalent of the formula represented by the trie. If there is no possibility of confusion, we will often use the term branch for the clause it represents. In general, the CNF formula can be significantly larger than the corresponding trie. Tries that represent logical formulas can be interpreted directly as formulas in negation normal form (NNF): A trie consisting of a single node represents the label of that node. Otherwise, the trie represents the disjunction of the label of the root with the conjunction of the formulas represented by the tries rooted at its children.

In this paper, we will assume that a variable ordering has been selected, and that nodes along a branch are labeled consistently with that ordering. A trie that stores all (non-tautological) implicates of a formula is called a *complete implicate trie*. For a formal definition and its properties, see [11].

Recall that for any logical formulas $\mathcal{F}$ and α and subformula $\mathcal{G}$ of $\mathcal{F}$, $\mathcal{F}[\alpha/\mathcal{G}]$ denotes the formula produced by substituting α for every occurrence of $\mathcal{G}$ in $\mathcal{F}$. If α is a truth functional constant 0 or 1 ($false$ or $true$), and if p is a negative literal, we will slightly abuse this notation by interpreting the substitution $[0/p]$ to mean that 1 is substituted for the atom that p negates.

The following simplification rules, when applied to a complete implicate trie, will produce a *reduced implicate trie* (ri-trie).

[1] The term *clause* is also used for a conjunction of literals, especially with *disjunctive normal form*.

[2] Many variations have been proposed in which arcs rather than nodes are labeled, and the labels are sometimes strings rather than single symbols.

$$\begin{array}{llll}\textbf{SR1}. & \mathcal{F} \rightarrow \mathcal{F}[\mathcal{G}/\mathcal{G} \vee 0] & \mathcal{F} \rightarrow \mathcal{F}[\mathcal{G}/\mathcal{G} \wedge 1] \\ \textbf{SR2}. & \mathcal{F} \rightarrow \mathcal{F}[0/\mathcal{G} \wedge 0] & \mathcal{F} \rightarrow \mathcal{F}[1/\mathcal{G} \vee 1] \\ \textbf{SR3}. & \mathcal{F} \rightarrow \mathcal{F}[0/p \wedge \neg p] & \mathcal{F} \rightarrow \mathcal{F}[1/p \vee \neg p]\end{array}$$

The branches of an ri-trie represent the *relatively prime implicates* [11]: If $\mathcal{F}$ is a logical formula, and if the variables of $\mathcal{F}$ are ordered, then a relatively prime implicate is one for which no proper prefix is also an implicate. If the leaf node of a branch in an ri-trie is labeled p_i, then every extension with variables of index greater than i is a branch in the complete implicate trie of $\mathcal{F}$. These extensions correspond to implicates of $\mathcal{F}$ that are not relatively prime and that are represented implicitly by that branch in the ri-trie.

Theorem 1. Given a logical formula $\mathcal{F}$ and an ordering of the variables of $\mathcal{F}$, then the branches of the corresponding ri-trie represent precisely the relatively prime implicates. In particular, the prime implicates are relatively prime, and each is represented by a branch in the trie. □

2.2 Computing Reduced Implicate Tries

Let $\mathcal{F}$ be a logical formula, and let the variables of $\mathcal{F}$ be $V = \{v_1, v_2, ..., v_n\}$. Then the ri-trie of $\mathcal{F}$ can be obtained by applying the recursively defined RIT operator (introduced in [10]):

$$\mathrm{RIT}(\mathcal{F}, V) = \begin{cases} \mathcal{F} & V = \emptyset \\ \left(\begin{array}{c} v_i \ \vee \ \mathrm{RIT}(\mathcal{F}[0/v_i], V - \{v_i\}) \\ \wedge \\ \neg v_i \ \vee \ \mathrm{RIT}(\mathcal{F}[1/v_i], V - \{v_i\}) \\ \wedge \\ \mathrm{RIT}((\mathcal{F}[0/v_i] \ \vee \ \mathcal{F}[1/v_i]), V - \{v_i\}) \end{array}\right) & v_i \in V \end{cases}$$

where v_i is the variable of lowest index in V.

Implicit is the use of simplification rules **SR1**, **SR2**, and **SR3**.

Theorem 2. If $\mathcal{F}$ is a logical formula with variable set V, then $\mathrm{RIT}(\mathcal{F}, V)$ is logically equivalent to $\mathcal{F}$. □

Let $\mathrm{Imp}(\mathcal{F})$ denote the set of all implicates of $\mathcal{F}$.

Lemma 1. Given logical formulas $\mathcal{F}$ and $\mathcal{G}$, $\mathrm{Imp}(\mathcal{F}) \cap \mathrm{Imp}(\mathcal{G}) = \mathrm{Imp}(\mathcal{F} \vee \mathcal{G})$. □

The last lemma means that the implicates being computed in the third conjunct of the RIT operator are precisely those that occur in both of the first two (ignoring, of course, the root labels v_i and $\neg v_i$). This third conjunct can thus be computed from the first two, and the direct recursive call on $(\mathcal{F}[0/v_i] \ \vee \ \mathcal{F}[1/v_i])$ can be avoided. This is significant because in this call, the size of the argument essentially doubles.

Theorem 3. Let $\mathcal{F}$ be a logical formula with variable set V, and let C be an implicate of $\mathcal{F}$. Then there is a unique branch of $\mathrm{RIT}(\mathcal{F}, V)$ that is a prefix of C, and every branch is a relatively prime implicate. □

2.3 Ternary Representation

Observe that the RIT operator essentially produces a conjunction of three tries. It is therefore natural to represent an ri-trie as a ternary trie. The root of the third subtrie is labeled 0. One advantage of this representation is that the ith variable appears only at level i. Another is that any subtrie (including the entire trie) is easily expressed as a four-tuple consisting of its root and the three subtries. For example, for a subtrie $\mathcal{T}$ we might write $\langle r, \mathcal{T}^+, \mathcal{T}^-, \mathcal{T}^0 \rangle$, where r is the root label of $\mathcal{T}$, and $\mathcal{T}^+$, $\mathcal{T}^-$, and $\mathcal{T}^0$ are the three subtries.

A trivial technical difficulty arises with the ternary representation: The zeroes along branches interfere with the prefix property of Theorem 3. But this is easily dealt with by interpreting the statement, *A branch B is a prefix of a clause C,* to mean *The clause represented by B* **with zeroes simplified away** *is a prefix of C.* The zeroes cause no difficulty when traversing branches in the trie.

Obtaining the ternary representation with the RIT operator requires only a minor change: disjoining 0 to the third conjunct. The notation $ri(\mathcal{F}, V) = 0 \vee \mathrm{RIT}(\mathcal{F}, V)$ will be used for the ternary ri-trie of $\mathcal{F}$ with variable ordering V. For the remainder of this paper, we will generally assume this ternary representation. As a result, the forest denoted by $\mathrm{RIT}(\mathcal{F}, V)$ will contain three tries whose roots are labeled by a variable, its complement, and zero.

Theorem 4. Let $\mathcal{F}$ and $\mathcal{G}$ be logically equivalent formulas. Then, with respect to a fixed variable ordering V, $ri(\mathcal{F}, V)$ is isomorphic to $ri(\mathcal{G}, V)$. □

2.4 Intersecting ri-Tries

Given two formulas $\mathcal{F}$ and $\mathcal{G}$, fix an ordering of the union of their variable sets, and let $\mathcal{T}_\mathcal{F}$ and $\mathcal{T}_\mathcal{G}$ be the corresponding ri-tries. The *intersection* of $\mathcal{T}_\mathcal{F}$ and $\mathcal{T}_\mathcal{G}$ is defined to be the ri-trie (with respect to the given variable ordering) that represents the intersection of the implicate sets. By Theorems 2 and 4 and Lemma 1, this is the ri-trie for $\mathcal{F} \vee \mathcal{G}$.

The intersection of two tries (with the same variable ordering) is produced by the INT operator introduced in [12].

Theorem 5. Let $\mathcal{T}_\mathcal{F}$ and $\mathcal{T}_\mathcal{G}$ be the respective ri-tries of $\mathcal{F}$ and $\mathcal{G}$ (with the same variable ordering). Then $\mathrm{INT}(\mathcal{T}_\mathcal{F}, \mathcal{T}_\mathcal{G})$ is the intersection of $\mathcal{T}_\mathcal{F}$ and $\mathcal{T}_\mathcal{G}$; in particular, $\mathrm{INT}(\mathcal{T}_\mathcal{F}, \mathcal{T}_\mathcal{G})$ is the ri-trie of $\mathcal{F} \vee \mathcal{G}$ (with respect to the given variable ordering). □

Theorem 5 provides a formal basis for a definition of the RIT operator that produces ri-tries using intersection. It is obtained from the earlier definition essentially by replacing the third conjunct by $\mathrm{INT}(\mathrm{RIT}(\mathcal{F}[0/v_i], V - \{v_i\}),\ \mathrm{RIT}(\mathcal{F}[1/v_i], V - \{v_i\}))$.

3 Prime Implicate Tries

Let $\mathcal{F}$ be a formula and let $\mathcal{T}_{\mathcal{F}}$ be the ri-trie for $\mathcal{F}$ under variable ordering V. From Theorem 3, there is a 1-1 correspondence between relatively prime implicates of $\mathcal{F}$ and branches in $\mathcal{T}_{\mathcal{F}}$. The sub-trie of $\mathcal{T}_{\mathcal{F}}$ consisting of the branches that correspond to prime implicates, which are relatively prime, is called a *prime implicate trie*, or pi-trie. As with ri-tries, it is convenient to use the ternary notation with pi-tries. The pi-trie for $\mathcal{F}$ is denoted $pi(\mathcal{F})$, and the set of prime implicates is denoted $\mathcal{P}(\mathcal{F})$. If B is a branch in $pi(\mathcal{F})$ whose labels form the implicate $\{p_1, \ldots, p_n\}$, a *suffix* of B is a sub branch having the labels $\{p_j, p_{j+1}, \ldots, p_n\}$, $1 \leq j \leq n+1$. This suffix is referred to as the suffix of B beginning with p_j; note that if $j = n+1$, the suffix is empty.

The algorithm presented in Section 3.2 was developed from the structure of ri-tries, but the algorithm does not compute pi-tries from ri-tries. To do so would be potentially inefficient, because the ri-trie may be considerably larger than the pi-trie. The approach adopted here computes the pi-trie directly. Not only can the resulting pi-trie be smaller than its ri-trie counterpart, but the 'intermediate bulge' that must be surmounted in both computations can also be smaller for the pi-trie.

3.1 The Structure of Prime Implicate Tries

The results in this subsection are required to develop the pi-trie algorithm presented in the next subsection. It is interesting to note that dealing with constants is a bit tricky. A single node labeled 0 is the pi-trie of a contradiction. Thus all clauses are implicates, and the empty clause is the only prime implicate. A single node labeled 1, on the other hand, is the pi-trie of a tautology, and thus its implicate set is empty.

Lemma 2. Let $\mathcal{F}$ and $\mathcal{G}$ be logical formulas, and let $C \in \mathcal{P}(\mathcal{F} \vee \mathcal{G})$. Then there exist prime implicates C_0 of $\mathcal{F}$ and C_1 of $\mathcal{G}$ such that $C = C_0 \cup C_1$.

Proof. Let D_0 be the subset of C that has the variables of C that occur in $\mathcal{F}$; similarly let D_1 be the subset of C that has the variables of C that are in $\mathcal{G}$.

First, D_0 is an implicate of $\mathcal{F}$ (and D_1 is an implicate of $\mathcal{G}$): If I is an interpretation such that $I(\mathcal{F}) = 1$, extend I so that I falsifies all literals in $C - D_0$. Since $I(\mathcal{F}) = 1$, $I(\mathcal{F} \vee \mathcal{G}) = 1$, so $I(C) = 1$. Thus $I(D_0) = 1$.

Now let C_i be a prime subset of D_i ($i = 0, 1$), and consider $C_0 \cup C_1$. Since $C_0 \cup C_1 \subseteq C$, it suffices to prove that $C_0 \cup C_1$ is an implicate of $\mathcal{F} \vee \mathcal{G}$. Suppose $I(\mathcal{F} \vee \mathcal{G}) = 1$; then $I(\mathcal{F}) = 1$ or $I(\mathcal{G}) = 1$, say $I(\mathcal{F}) = 1$. Then, since C_0 is an implicate of $\mathcal{F}$, $I(C_0) = 1$, so $I(C_0 \cup C_1) = 1$. □

Note that C_0 and C_1 need not be disjoint. Note also that the lemma admits the possibility that C_0 or C_1 might be empty. But if that occurs, then the corresponding formula must be unsatisfiable.

Suppose that formulas $\mathcal{F}$ and $\mathcal{G}$ have, respectively, pi-tries $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{G}}$. The branches of $\mathcal{T}_{\mathcal{F}}$ and of $\mathcal{T}_{\mathcal{G}}$ correspond to $\mathcal{P}(\mathcal{F})$ and to $\mathcal{P}(\mathcal{G})$. Define the *prime union* of $\mathcal{P}(\mathcal{F})$ and $\mathcal{P}(\mathcal{G})$ to be

$$\{C_0 \cup C_1 | C_0 \in \mathcal{P}(\mathcal{F}), C_1 \in \mathcal{P}(\mathcal{G})\}$$

Lemma 2 assures that $\mathcal{P}(\mathcal{F} \vee \mathcal{G})$ is a subset of the prime union.

Lemma 3. Let $\mathcal{F}$ be a logical formula, and let p be a literal whose variable does not occur in $\mathcal{F}$. If C is an implicate of $\mathcal{F}$ not containing the variable of p, then C is not an implicate of $p \vee \mathcal{F}$.

Proof. Let I falsify C and satisfy p; then I satisfies $p \vee \mathcal{F}$ but not C. □

Lemma 4. Let $\mathcal{F}$ be a logical formula, and let p be a literal whose variable does not occur in $\mathcal{F}$. Then C is an implicate of $\mathcal{F}$ iff $\{p\} \cup C$ is an implicate of $p \vee \mathcal{F}$.

Proof. Suppose first that C is an implicate of $\mathcal{F}$; it suffices to show that any interpretation I that falsifies $\{p\} \cup C$ falsifies $p \vee \mathcal{F}$. Since I falsifies C, I falsifies $\mathcal{F}$, and, in turn, since I falsifies p, I falsifies $p \vee \mathcal{F}$.

Now assume that $\{p\} \cup C$ is an implicate of $p \vee \mathcal{F}$, and consider a satisfying interpretation I for $\mathcal{F}$. Extend I to falsify p. Since I satisfies $\mathcal{F}$, I satisfies $(p \vee \mathcal{F})$, so I satisfies $\{p\} \cup C$. Since $I(p) = 0$, C is satisfied. □

This result can be extended to prime implicates.

Lemma 5. Let $\mathcal{F}$ be a logical formula, and let p be a literal whose variable does not occur in $\mathcal{F}$. Then C is a prime implicate of $\mathcal{F}$ iff $\{p\} \cup C$ is a prime implicate of $p \vee \mathcal{F}$.

Proof. Suppose first that C is a prime implicate of $\mathcal{F}$. By the previous lemma, $p \cup C$ is an implicate of $p \vee \mathcal{F}$. Consider a prime subset D of $\{p\} \cup C$. Then D must contain p by Lemma 3, say $D = \{p\} \cup D'$. We must show that $D' = C$. But this is immediate since, by the previous lemma, D' is an implicate of $\mathcal{F}$.

Now suppose that $\{p\} \cup C$ is a prime implicate of $p \vee \mathcal{F}$. Then it is immediate from the previous lemma that C is a prime implicate of $\mathcal{F}$. □

Lemma 6. Let $\mathcal{F}$ be a logical formula and p a literal, and let $\mathcal{F}_0 = \mathcal{F}[0/p]$. Then $\{p\} \cup C$ is an implicate of $\mathcal{F}$ iff C is an implicate of $\mathcal{F}_0$. Moreover, if $\{p\} \cup C$ is a prime implicate of $\mathcal{F}$, $p \notin C$, then C is a prime implicate of $\mathcal{F}_0$. The analogous results for $\mathcal{F}_1 = \mathcal{F}[1/p]$ are also valid.

Proof. Suppose first that $\{p\} \cup C$ is an implicate of $\mathcal{F}$, and let I be an interpretation satisfying $\mathcal{F}_0$. Extend I to falsify p. Then by the definition of $\mathcal{F}_0$, $I(\mathcal{F}) = 1$. So I satisfies $\{p\} \cup C$ but falsifies p, so I satisfies C.

Now assume that C is an implicate of $\mathcal{F}_0$, and let I be a satisfying interpretation for $\mathcal{F}$. If $I(p) = 1$, then $I(\{p\} \cup C) = 1$. If $I(p) = 0$, then $I(\mathcal{F}_0) = 1$, so $I(C) = 1 = I(\{p\} \cup C)$.

Finally, suppose that $\{p\} \cup C$ is a prime implicate of $\mathcal{F}$. Then, by the first part of the Lemma, C is an implicate of $\mathcal{F}_0$. If a subset D of C is also an implicate, we must show that $D = C$. Then $\{p\} \cup D$ is an implicate of $\mathcal{F}$. Since $\{p\} \cup D$ is a subset of $\{p\} \cup C$, and since $\{p\} \cup C$ is prime, $\{p\} \cup D = \{p\} \cup C$ and $D = C$.

The proof for $\mathcal{F}_1$ is entirely similar. □

The reader is reminded that, for a logical formula $\mathcal{F}$, $\mathcal{F} \equiv (p \vee \mathcal{F}_0) \wedge (\overline{p} \vee \mathcal{F}_1)$, where $\mathcal{F}_0 = \mathcal{F}[0/p]$ and $\mathcal{F}_1 = \mathcal{F}[1/p]$. The next lemma characterizes a clause C and and a formula $\mathcal{F}$ when C is a prime implicate of $\mathcal{F}_0$ but not of $\mathcal{F}$ itself.

Lemma 7. Let $\mathcal{F}$ be a logical formula and p a literal, let $\mathcal{F}_0 = \mathcal{F}[0/p]$ and $\mathcal{F}_1 = \mathcal{F}[1/p]$, and let C be a clause with $p \notin C$. Then C is a prime implicate of $\mathcal{F}_0$ and $\{p\} \cup C$ is not a prime implicate of $\mathcal{F}$ iff C is prime for $\mathcal{F}$ and an implicate of $\mathcal{F}_1$. In that case, if D is a subset of C and a prime implicate of $\mathcal{F}_1$, and if $C \neq D$, then $\{\overline{p}\} \cup D \in \mathcal{P}(\mathcal{F})$. The analogous result for a prime implicate of $\mathcal{F}_1$ is also valid.

Proof. Note that the variable of p does not occur in $\mathcal{F}_0$ or in $\mathcal{F}_1$. Assume first that C is prime for $\mathcal{F}_0$ and that $\{p\} \cup C$ is not prime for $\mathcal{F}$. By Lemma 5, $\{p\} \cup C$ is prime for $p \vee \mathcal{F}_0$, and by Lemma 6, it is an implicate of $\mathcal{F}$. So some proper subset D of $\{p\} \cup C$ is a prime implicate of $\mathcal{F}$. Also from Lemma 6, this prime subset cannot include p and is therefore a subset of C. Clearly, $p \vee D$ is an implicate of $\mathcal{F}$, thus D is an implicate of $\mathcal{F}_0$. So $D = C$ because C is prime for $\mathcal{F}_0$.

Now consider an interpretation I that falsifies C and satisfies p. Since C is an implicate of $\mathcal{F}$, $I(\mathcal{F}) = 0$, and since $I(p) = 1$, $I(p \vee \mathcal{F}_0) = 1$. Since $\mathcal{F} = (p \vee \mathcal{F}_0) \wedge (\overline{p} \vee \mathcal{F}_1)$, I must falsify $\overline{p} \vee \mathcal{F}_1$. Therefore I falsifies $\mathcal{F}_1$, making C an implicate of $\mathcal{F}_1$.

Assume now that C is prime for $\mathcal{F}$ and an implicate of $\mathcal{F}_1$. Let I be an interpretation that falsifies $\{p\} \cup C$. Since C is an implicate of both $\mathcal{F}_1$ and $\mathcal{F}$, $I(\mathcal{F}_1) = I(\mathcal{F}) = 0$. But $I(\overline{p} \vee \mathcal{F}_1) = 1$, so $I(p \vee \mathcal{F}_0) = 0$; as a result, $I(\mathcal{F}_0) = 0$, and C is an implicate of $\mathcal{F}_0$.

Finally, if $C \neq D$ — i.e., if D is a proper subset of C — we must show that $\{\overline{p}\} \cup D \in \mathcal{P}(\mathcal{F})$. But this is immediate from the first part of the lemma: Were $\{\overline{p}\} \cup D \notin \mathcal{P}(\mathcal{F})$, D would be an implicate of $\mathcal{F}_0$ and a proper subset of a prime implicate of $\mathcal{F}_0$. □

The next example illustrates the previous five lemmas, which relate the implicates of a formula $\mathcal{F}$ to those of $\mathcal{F}_0$ and of $\mathcal{F}_1$.

Example. Let

$$\mathcal{F} = ((p \vee (q \wedge r)) \wedge (\overline{p} \vee \overline{q} \vee r)).$$

The set of prime implicates of $\mathcal{F}$ is

$$\mathcal{P}(\mathcal{F}) = \{\{p, q\}, \{p, r\}, \{\overline{q}, r\}\},$$

and

$$\mathcal{F}_0 = \mathcal{F}[0/p] = (q \wedge r) \quad \text{and} \quad \mathcal{F}_1 = \mathcal{F}[1/p] = (\overline{q} \vee r).$$

Consider Lemmas 3 and 4. Let $C = \{p, q, r\}$. Then $C \in \mathrm{Imp}(\mathcal{F})$, the variable s does not occur in $\mathcal{F}$, and $s \notin C$. Lemma 3 assures us that $C \notin \mathrm{Imp}(s \vee \mathcal{F})$, and Lemma 4 that $C \cup \{s\} \in \mathrm{Imp}(s \vee \mathcal{F})$. Lemma 5 extends Lemma 4 to prime implicates. To illustrate, now let $C = \{\overline{q}, r\}$. Then $C \in \mathcal{P}(\mathcal{F})$, and $C \cup \{s\} \in \mathcal{P}(s \vee \mathcal{F})$. For Lemma 6, let $C = \{q\}$. Then $C \in \mathcal{P}(\mathcal{F}_0)$ and $C \cup \{p\} \in \mathcal{P}(\mathcal{F})$. Letting $C = \{q, r\}$ illustrates Lemma 6 for the non-prime case.

Lemma 7 is probably the one in most need of explanation. This example illustrates the dual: If $C = \{\overline{q}, r\}$, then $C \in \mathcal{P}(\mathcal{F}_1)$ and $C \cup \{\overline{p}\} \notin \mathcal{P}(\mathcal{F})$. The lemma guarantees that $C \in \mathcal{P}(\mathcal{F})$ and that $C \in \mathrm{Imp}(\mathcal{F}_0)$. Since C is a proper superset of a prime implicate $D = \{r\}$ of $\mathcal{F}_0$, the lemma also guarantees that $\{p\} \cup D \in \mathcal{P}(\mathcal{F})$, which is indeed the case.

Lemma 8. If $\mathcal{F}$ is satisfiable and $\mathcal{F}_0$ is unsatisfiable, then $\mathcal{P}(\mathcal{F}) = \{\{p\}\} \cup \mathcal{P}(\mathcal{F}_1)$. Similarly, if $\mathcal{F}_1$ is unsatisfiable, then $\mathcal{P}(\mathcal{F}) = \{\{\overline{p}\}\} \cup \mathcal{P}(\mathcal{F}_0)$. In particular, if the unit $\{p\}$ is an implicate of $\mathcal{F}$, then no prime implicate of $\mathcal{F}$ contains $\overline{p}$.

Proof. Suppose interpretation I satisfies $\mathcal{F}$. Then, since $\mathcal{F}_0$ is unsatisfiable, $I(p) = 1$. Thus the unit $\{p\}$ is a prime implicate.

On the other hand, any interpretation that satisfies $\mathcal{F}_1$ extends to an interpretation that satisfies $\mathcal{F}$ by assigning $p = 1$. Thus the implicates of $\mathcal{F}$ not containing p are exactly the implicates of $\mathcal{F}_1$ that do not contain p. □

Lemma 9. If $\mathcal{F}$ is satisfiable and $\mathcal{F}_0$ is a tautology, then $\mathcal{P}(\mathcal{F}) = \{\{\neg p\} \cup C \mid C \in \mathcal{P}(\mathcal{F}_1)\}$. Similarly, if $\mathcal{F}_1$ is a tautology, then $\mathcal{P}(\mathcal{F}) = \{\{p\} \cup C \mid C \in \mathcal{P}(\mathcal{F}_0)\}$.

Proof. First note that every implicate must contain $\overline{p}$, for if C is a clause that does not contain $\overline{p}$, let I be the interpretation that assigns 1 to $\overline{p}$ and 0 to all other literals in C. Then I satisfies $\mathcal{F}$ but not C. Also, by Lemma 7, since $\mathcal{F}_0$ has no implicates, if C is a prime implicate of $\mathcal{F}_1$, then $\{\overline{p}\} \cup C$ is a prime implicate of $\mathcal{F}$. □

The next theorem summarizes the parts of the lemmas that are explicitly used in the algorithm in the next section.

Theorem 6. Let $\mathcal{F}$ be a logical formula and p a literal, let $\mathcal{F}_0 = \mathcal{F}[0/p]$, let $\mathcal{F}_1 = \mathcal{F}[1/p]$, and suppose $C \in \mathcal{P}(\mathcal{F}_0)$ and $D \in \mathcal{P}(\mathcal{F}_1)$. Then

1. If $C = D$, then $C \in \mathcal{P}(\mathcal{F})$ (and thus $\{p\} \cup C \notin \mathcal{P}(\mathcal{F})$).
2. If $C \subset D$ (i.e., proper subset), then $D \in \mathcal{P}(\mathcal{F})$ and $\{p\} \cup C \in \mathcal{P}(\mathcal{F})$.
3. If no subset relationship exists between C and D, then a subset of $C \cup D$ is a prime implicate of $\mathcal{F}$. □

Observe that if the literal p is redundant or does not occur in $\mathcal{F}$, then only Case 1 of the theorem applies.

3.2 An Algorithm for Prime Implicate Tries

The algorithm developed here is not entirely transparent; every effort has been made to explain how it works. The algorithm builds the pi-trie of a logical formula recursively, processing one variable at a time. If $\mathcal{F}$ is the formula and p the variable being processed, let $\mathcal{F}_0 = \mathcal{F}[0/p]$ and $\mathcal{F}_1 = \mathcal{F}[1/p]$, as before. The base case is a constant, and the pi-trie is a single node labeled with the constant. The procedure **prime** handles the base case and makes the recursive calls, each of which reduces the number of variables by at least 1.[3]

The output of the algorithm is a ternary pi-trie. The first subtrie contains all prime implicates of $\mathcal{F}$ that contain p, the second has all prime implicates that contain $\neg p$, and the third all prime implicates that contain neither p nor $\neg p$. The function **prime** initializes the first subtrie to $p \vee pi(\mathcal{F}_0)$ (recall that $pi(\mathcal{F}_0)$ is the prime implicate trie of $\mathcal{F}_0$), and the second to $\neg p \vee pi(\mathcal{F}_1)$. Both are supersets of the desired tries. Repeated

[3] Substituting a truth constant for one variable and simplifying may remove other variables.

applications of Theorem 6 removes redundant branches from the first two subtries while building the third, which is the pi-trie for $\mathcal{F}_0 \vee \mathcal{F}_1$. The **PIT** procedure builds the third subtrie from the first two; in the process, all unnecessary branches are removed from the first two. The procedure **copycheck** prunes non-minimal branches from the third sub-trie.

The description below is intended to assist the reader with the details of the pi-trie algorithm. The comments in steps are references to lemmas and theorems that justify the steps. The reader is reminded that tautologies have no implicates, and hence, $pi(1)$ is a single node labeled 1 and represents the empty set. On the other hand, all clauses are implicates of a contradiction. Thus, the empty clause is the only prime implicate, and $pi(0)$ is a single node labeled 0 and represents $\{\Box\}$.

Algorithm Description

1. The base case is a constant; the pi-trie is a single node labeled with the constant.
2. If p is the variable being processed in a recursive call, initialize $\mathcal{T}_0$ to $pi(\mathcal{F}_0)$, $\mathcal{T}_1$ to $pi(\mathcal{F}_1)$, and $\mathcal{T}_2$ to empty.
3. Apply Theorem 6 to each pair of branches C in $\mathcal{T}_0$ and D in $\mathcal{T}_1$. Note that *move* means *remove* and *place*. Note also that moving C means that it will never be paired up again in this loop.
   (a) If $C = D$, then move C to $\mathcal{T}_2$ and remove D from $\mathcal{T}_1$. {Theorem 6, Part 1}
   (b) If $C \subset D$ (proper subset), then
      i. Move D to $\mathcal{T}_2$ and mark it prime. {Theorem 6, Part 2}
      ii. Mark C prime.
   (c) Else-if $D \subset C$, then
      i. Move C to $\mathcal{T}_2$ and mark it prime. {Theorem 6, Part 2}
      ii. Mark D prime.
   (d) Else (i.e., no subset relationship between C and D) {Theorem 6, Part 3}
   Form $C \cup D$ and move to $\mathcal{T}_2$.
4. After all pairs of branches have been processed {Theorem 6}
   (a) If $\mathcal{T}_0$ (or $\mathcal{T}_1$) is a leaf, set it to $\emptyset$, since p is not an implicate of $(p \vee \mathcal{F}_0)$.
   (b) $p \vee \mathcal{T}_0 = pi(p \vee \mathcal{F}_0)$ and $\neg p \vee \mathcal{T}_1 = pi(\neg p \vee \mathcal{F}_1)$.
   (c) The branches of $\mathcal{T}_2$ are implicates of $(\mathcal{F}_0 \vee \mathcal{F}_1)$ and $\mathcal{T}_2 \supset pi(\mathcal{F}_0 \vee \mathcal{F}_1)$.
5. Remove any branch in $\mathcal{T}_2$ that is a superset of another branch.
6. The routine **PIT** below handles Steps 3 and 5 simultaneously.
7. After all pairs of branches in $\mathcal{T}_2$ have been processed in Step 5, the pi-trie for $\mathcal{F}$ is (subject to simplification)
$$pi(\mathcal{F}) = 0 \vee [(p \vee \mathcal{T}_0) \wedge (\neg p \vee \mathcal{T}_1) \wedge \mathcal{T}_2)].$$
8. Simplification is necessary if $\mathcal{T}_0$ or $\mathcal{T}_1$ is a a single node labeled with a constant.
   (a) $\mathcal{T}_0 = 0, \mathcal{T}_1 = 0$. Then $pi(\mathcal{F}) = pi(p \wedge \neg p) = 0$.
   (b) $\mathcal{T}_0 = 0, \mathcal{T}_1 = 1$. Then $\mathcal{T}_2 = 1$ and $pi(\mathcal{F}) = pi((p \vee 0) \wedge 1) = p$.
   (c) $\mathcal{T}_0 = 0, \mathcal{T}_1$ is not constant. {Lemma 8}
      i. $p \vee \mathcal{T}_0 = p$, so the unit p is a prime implicate.
      ii. Resolving p with $\neg p \vee \mathcal{T}_1$ produces $\mathcal{T}_1$.
      iii. $\mathcal{T}_2 = \mathcal{T}_1$ and there are no prime implicates containing $\neg p$.
      iv. $pi(\mathcal{F}) = 0 \vee (p \wedge \mathcal{T}_2)$.

(d) $\mathcal{T}_0 = 1, \mathcal{T}_1 = 0$ is similar to the case $\mathcal{T}_0 = 0, \mathcal{T}_1 = 1$.
(e) $\mathcal{T}_0 = 1, \mathcal{T}_1 = 1$. Then $pi(\mathcal{F}) = pi(0 \vee 1) = 1$.
(f) $\mathcal{T}_0 = 1, \mathcal{T}_1$ is not constant. {Lemma 9}
  i. $\mathcal{T}_0 = 1$ since tautologies have no implicates.
  ii. $p \vee \mathcal{T}_0 = 1$, so $pi(p \vee \mathcal{T}_0) = 1$.
  iii. $\mathcal{F}_0 \vee \mathcal{F}_1 \equiv 1$, so $\mathcal{T}_2 = 1$.
  iv. $pi(\mathcal{F}) = 0 \vee (\neg p \vee \mathcal{T}_1)$.
(g) There are similar cases when $\mathcal{T}_1$ is constant and $\mathcal{T}_0$ is not.

This description together with the results from Section 3.1 prove

Theorem 7. If a logical formula $\mathcal{F}$ is input to the pi-trie algorithm, then the algorithm terminates and produces $pi(\mathcal{F})$.

Algorithm Pseudocode

The pi-trie algorithm is initialized with the call **prime**$(0, \mathcal{F}, 1)$; **prime** handles the base cases — truth constants — and calls the procedure **PIT**. In turn, **PIT** calls the routine, **copycheck**, which purges non-prime branches in the third sub-trie. The routine **prime** is pseudocode for Steps 7 and 8 in the algorithm description. Note that $\mathcal{T}_0^*$ is merely $\mathcal{T}_0$ but with its root set to zero; similiarly for $\mathcal{T}_1^*$. These are needed in the two cases in which non-constant tries originally computed as first or second subtries must be installed as a third subtrie and thus must have a zero root.

The primary functions of the routine **PIT** are to remove redundant branches from the subtries $\mathcal{T}_0$ and $\mathcal{T}_1$ rooted at, respectively, p and $\neg p$, and to initialize the third subtrie, $\mathcal{T}_2$, rooted at 0. The routine is a straightforward iteration over all branch pairs consisting of one branch from $\mathcal{T}_0$ and one from $\mathcal{T}_1$. Theorem 6 describes which branches need to be moved to $\mathcal{T}_2$ and which remain.

The intricate control structure in **PIT** is there to reduce the number of iterations and the number of subset checks. The comments in the routine refer to the part of Theorem 6 that justifies the particular control. For example, if a subset relation between branches is discovered — say C is in $\mathcal{T}_0$, D is in $\mathcal{T}_1$, and $C \subset D$ — then Part 2 of the theorem tells us that C remains in $\mathcal{T}_0$ and that D should be moved to $\mathcal{T}_2$. Each can then be marked *prime*, and neither can be a superset of another implicate, potentially eliminating subset checks. Note that whenever all branches of the form b_i are removed from $\mathcal{T}_i$, $i = 0, 1$, then $\mathcal{T}_i$ is a root leaf and must be set to $\emptyset$.

Part 3 of the theorem applies when no subset relationship exists between branches. The union is installed in $\mathcal{T}_2$, and the **copycheck** routine is called to determine whether the union is in fact a branch in $\mathcal{T}_2$.

The purpose of the routine **copycheck** is to remove redundant branches from the third subtrie $\mathcal{T}_2$. The first parameter b is the branch passed to it, and the second, $\mathcal{T}$, is the subtrie $\mathcal{T}_2$. If b is a tautology, which is possible when it is the result of a union, or if a prefix of b is already in $\mathcal{T}$, b does not belong in $\mathcal{T}$, and thus there is nothing to be done by the routine.

The third parameter is the boolean $testflag$. Whenever it is known that no branch in $\mathcal{T}$ can subsume b, $testflag = 0$. Thus, if the passed branch is marked prime, 0 is passed to $testflag$; otherwise 1 is passed. In the routine itself, if it is discovered that b is a subset of a branch $\tilde{b}$, then $\tilde{b}$ is removed. No other branch in $\mathcal{T}$ can be a subset of $\tilde{b}$ — otherwise, $\tilde{b}$ would already have been removed — so no branch in $\mathcal{T}$ can be a subset of b. Thus $testflag$ can be set to 0.

```
definefunction prime ($root$:TrieNode, $\mathcal{F}$:LogicalFormula, $i$:index):Trie;
local TrieNode $\mathcal{T}_0, \mathcal{T}_1$;
if $\mathcal{F}$ is constant then return ($\mathcal{F}$);
$\mathcal{T}_0 \longleftarrow$ prime($p_i, \mathcal{F}[0/p_i], i+1$);    $\mathcal{T}_0^* \longleftarrow \mathcal{T}_0[0/p_i]$;
$\mathcal{T}_1 \longleftarrow$ prime($\neg p_i, \mathcal{F}[1/p_i], i+1$);    $\mathcal{T}_1^* \longleftarrow \mathcal{T}_1[0/\neg p_i]$;
switch $\mathcal{T}_0, \mathcal{T}_1$ do
    case ($\mathcal{T}_0, \mathcal{T}_1$ are not constant) return (PIT ($root, \mathcal{T}_0, \mathcal{T}_1, 1$));
    case $\mathcal{T}_0 = 0 \wedge \mathcal{T}_1 = 0$ return (0);
    case $\mathcal{T}_0 = 0 \wedge \mathcal{T}_1 = 1$ return ($\langle root, p_i, 1, 1\rangle$);
    case $\mathcal{T}_0 = 0 \wedge (\mathcal{T}_1$ is not constant$)$ return ($\langle root, p_i, 1, \mathcal{T}_1^*\rangle$);
    case $\mathcal{T}_0 = 1 \wedge \mathcal{T}_1 = 0$ return ($\langle root, 1, \neg p_i, 1\rangle$);
    case $\mathcal{T}_0 = 1 \wedge \mathcal{T}_1 = 1$ return (1);
    case $\mathcal{T}_0 = 1 \wedge (\mathcal{T}_1$ is not constant$)$ return ($\langle root, 1, \mathcal{T}_1, 1\rangle$);
    case $(\mathcal{T}_0$ is not constant$) \wedge \mathcal{T}_1 = 0$ return ($\langle root, 1, \neg p_i, \mathcal{T}_0^*\rangle$);
    case $(\mathcal{T}_0$ is not constant$) \wedge \mathcal{T}_1 = 1$ return ($\langle root, \mathcal{T}_0, 1, 1\rangle$);
end
end prime;
```

To illustrate how parts of the algorithm work, consider again the example from Section 3.1:

$$\mathcal{F} = ((p \vee (q \wedge r)) \wedge (\overline{p} \vee \overline{q} \vee r)).$$

The set of prime implicates of $\mathcal{F}$ is $\mathcal{P}(\mathcal{F}) = \{\{p, q\}, \{p, r\}, \{\overline{q}, r\}\}; \mathcal{F}_0 = \mathcal{F}[0/p] = (q \wedge r)$, and $\mathcal{F}_1 = \mathcal{F}[1/p] = (\overline{q} \vee r)$. Assume the variables are in the order p, q, r.

The first two subtries of $\mathcal{T}$ will be denoted $\mathcal{T}_0$ and $\mathcal{T}_1$, respectively, which matches their local names in the initial invocation of **prime**. In the recursive invocation computing $\mathcal{T}_0$, the tries assigned locally to $\mathcal{T}_0$ and $\mathcal{T}_1$ will be denoted globally as $\mathcal{T}_{00}$ and $\mathcal{T}_{01}$. In general, a given subtrie will be identified by the branch leading to it corresponding to a string subscript of $\mathcal{T}$.

The function **prime** calls itself recursively until the formula input to it is constant. It determines $\mathcal{T}_0$ and $\mathcal{T}_1$ (locally) and invokes **PIT** precisely when both of these are non-constant. In the example, $\mathcal{T}_0 = \mathbf{prime}(p, (q \wedge r), 2)$ and $\mathcal{T}_1 = \mathbf{prime}(\neg p, (\neg q \vee r), 2)$. Consider $\mathcal{T}_0$, which is built entirely by **prime**. Then $\mathcal{T}_{00} = \mathbf{prime}(q, 0, 3) = 0$ and $\mathcal{T}_{01} = \mathbf{prime}(\neg q, r, 3)$.

Next, $\mathcal{T}_{010} = \mathbf{prime}(r, 0, 4) = 0$, and $\mathcal{T}_{011} = \mathbf{prime}(\neg r, 1, 4) = 1$. This triggers the third case in computing $\mathcal{T}_{01}$, and the result is $\langle \neg q, r, 1, 1\rangle$. Since $\mathcal{T}_{00} = 0$, the fourth case is triggered in the computation of $\mathcal{T}_0$. The result is that the version of $\mathcal{T}_{01}$ referenced in the invocation locally as $\mathcal{T}_1^*$ is installed as the third subtrie of $\mathcal{T}_0$. More precisely, $\mathcal{T}_0 = \langle p, q, 1, \langle 0, r, 1, 1\rangle\rangle$.

```
definefunction PIT ($root$:TrieNode, $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2$:Trie):Trie;
;/* $\mathcal{T}_0$, $\mathcal{T}_1$ are the $pi$-tries for $\mathcal{F}_0$, $\mathcal{F}_1$, respectively        */
;/* $\subset$ means proper subset                                                  */
foreach pair of branches $\tilde{b}_0, \tilde{b}_1 (\tilde{b}_i \in \mathcal{T}_i)$ do
    $b_i \longleftarrow \tilde{b}_i - \{root(\mathcal{T}_i)\}$;
    switch $b_0, b_1$ do
        case both are marked prime copycheck ($b_0 \cup b_1, \mathcal{T}_2, 1$);
        case $b_0$ is marked prime            /* no subsets of $b_0$ are in $\mathcal{T}_1$ */
            if $b_0 \subset b_1$ then                               /* Theorem 6 */
                MOVE $b_1$ to $\mathcal{T}_2$ and mark $b_1$ prime              /* Part 2 */;
                copycheck ($b_1, \mathcal{T}_2, 0$)
            else
                copycheck ($b_0 \cup b_1, \mathcal{T}_2, 1$)                   /* Part 3 */
            end
        end
        case $b_1$ is marked prime            /* no subsets of $b_1$ are in $\mathcal{T}_0$ */
            if $b_1 \subset b_0$ then                               /* Theorem 6 */
                MOVE $b_0$ to $\mathcal{T}_2$ and mark $b_0$ prime              /* Part 2 */;
                copycheck ($b_0, \mathcal{T}_2, 0$)
            else
                copycheck ($b_0 \cup b_1, \mathcal{T}_2, 1$)                   /* Part 3 */
            end
        end
        otherwise
            if $b_0 \subset b_1$ then                               /* Theorem 6 */
                MOVE $b_1$ to $\mathcal{T}_2$, mark $b_1$ and $b_0$ prime       /* Part 2 */;
                copycheck ($b_1, \mathcal{T}_2, 0$)
            else
                if $b_1 = b_0$ then                                 /* Theorem 6 */
                    DELETE $b_0$, MOVE $b_1$ to $\mathcal{T}_2$, mark $b_1$ prime    /* Pt 1 */;
                    copycheck ($b_1, \mathcal{T}_2, 0$)
                else
                    if $b_1 \subset b_0$ then                       /* Theorem 6 */
                        MOVE $b_0$ to $\mathcal{T}_2$, mark $b_0$ and $b_1$ prime   /* Part 2 */;
                        copycheck ($b_0, \mathcal{T}_2, 0$)
                    else
                        copycheck ($b_0 \cup b_1, \mathcal{T}_2, 1$)           /* Part 3 */
                    end
                end
            end
        end
    end
end
if $leaf(\mathcal{T}_1)$ then $\mathcal{T}_1 \leftarrow \emptyset$ else if $leaf(\mathcal{T}_2)$ then $\mathcal{T}_2 \leftarrow \emptyset$;
return ($\langle root, \mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2 \rangle$);
end PIT;
```

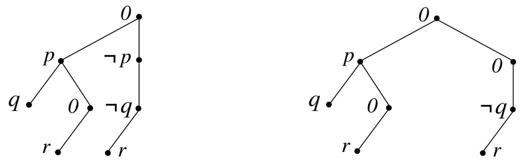


Fig. 1. Left: Building $\mathcal{T}_0$ and $\mathcal{T}_1$; Right: Final *pi*-trie

A similar analysis reveals that $\mathcal{T}_1 = \langle \neg p, 1, \langle \neg q, \langle r, 1, 1, 1\rangle, 1, 1\rangle, 1\rangle$. The upshot is that $\mathcal{T}_0$ has $\{p, q\}$ and $\{p, r\}$ as branches; $\mathcal{T}_1$ has only $\{\neg p, \neg q, r\}$. So in computing $\mathcal{T}$, **PIT** is called on these tries. This is the situation on the left in Figure 1.

When **PIT** examines the branch pair ($\{r\}$,$\{\neg q, r\}$), rooted at, respectively, p and $\neg p$, Part 2 of Theorem 6 applies. Since $\{\neg q, r\} \supset \{r\}$, $\{\neg q, r\}$ is placed in the zero subtrie of $\mathcal{T}$, and $\{p, r\}$ is marked prime (and not moved). This removes the only branch below the root $\neg p$ in $\mathcal{T}_1$. The iteration terminates, and $\mathcal{T}_1$ is set to $\emptyset$, as shown on the right in Figure 1.

```
define copycheck ($b, \mathcal{T}, testflag$);
if TAUTOLOGY(b) then EXIT;
if $b$ has a prefix in $\mathcal{T}$ then EXIT;
foreach branch $\tilde{b} \in \mathcal{T}$ do
    if $testflag$ then
        if $\tilde{b} \subseteq b$ then exit copycheck
    end
    if $\tilde{b}$ is not marked prime then
        if $b \subset \tilde{b}$ then
            remove $\tilde{b}$ from $\mathcal{T}$;
            $testflag \leftarrow 0$
        end
    end
end
add $b$ to $\mathcal{T}$;
end copycheck;
```

An example that illustrates the routine **copycheck** would be rather large and is not included due to space limitations.

3.3 Uniqueness of *pi*-Tries

Prime implicate tries can be represented as n-ary or as ternary trees, although in this paper attention has been restricted to the ternary representation. Reduced implicate tries are similar in this regard, and each representation is unique for ri-tries [11]: If $\mathcal{F}$ and $\mathcal{G}$ are logically equivalent, then $ri(\mathcal{F}, V)$ is isomorphic to $ri(\mathcal{G}, V)$. Formal proofs are provided in [11], but it is easy to see why this is true. Each branch represents a relatively prime implicate, so that, for a given logical formula, once the variable order has been chosen, the tree branches and the node labels are completely determined. The same is true for a pi-trie since the branches represent the prime implicates.

Definition. Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be directed acyclic graphs (dags). Then $\mathcal{D}_1$ and $\mathcal{D}_2$ are said to be *isomorphic* if there is a bijection f such that if (A, B) is an edge in $\mathcal{D}_1$, then $(f(A), f(B))$ is an edge in $\mathcal{D}_2$. If the nodes of the dags are labeled, then the isomorphism is *label-preserving* if for every node A, $Label(A) = Label(f(A))$.

Theorem 8. Let $\mathcal{F}$ and $\mathcal{G}$ be logically equivalent formulas. Then, with respect to a fixed variable ordering, $pi(\mathcal{F})$ is isomorphic to $pi(\mathcal{G})$. □

4 Preliminary Experiments

The algorithm from Section 3 has only recently been realized in a prototype implementation. Java was used for flexibility and to shorten development time. The prototype runs but is certainly in a "preliminary development" stage. The experiments compared the prototype to a simple resolution-based system. The results are mildly encouraging but do not support any conclusions about the potential efficiency of a pi-trie based system.

There were 11 sets of trials. Each set consisted of ten randomly generated CNF formulas for a fixed number of variables. The number of clauses in each set was four times the number of variables, and each clause contained 3 literals. The table shows the results for each set of trials.

Number of Variables	Number of Clauses	Number of Literals	Avg Number of Prime Implicates	Avg Time (msecs.)	
				prime	Resolution
7	28	84	14	29	58
8	32	96	6	12	200
9	36	108	13	31	459
10	40	120	15	138	1280
11	44	132	30	128	3906
12	48	144	36	286	8223
13	52	156	53	750	21470
14	56	168	47	1384	76465
15	60	180	49	3893	170757
16	64	192	63	3781	639090
17	68	204	48	1626	1124601

The first four columns list, for each trial set, the numbers of variables, clauses, and literals in each formula and the average number of prime implicates. Times are in the last two columns, given in milliseconds and are averaged over the ten formulas in each trial. Both algorithms were run on the same ten formulas. The pi-trie algorithm is consistently faster than the resolution-based algorithm: twice as fast in the smallest trial and 700 times as fast in the last. Although these results are very preliminary and inconclusive, they are enough to indicate pi-trie techniques are worthy of further development.

Acknowledgements

The authors are grateful for the useful and constructive comments from the reviewers.

References

1. Bittencourt, G.: Combining syntax and semantics through prime form representation. Journal of Logic and Computation 18, 13–33 (2008)
2. Coudert, O., Madre, J.: Implicit and incremental computation of primes and essential implicant primes of boolean functions. In: 29th ACM/IEEE Design Automation Conference, pp. 36–39 (1992)
3. de Kleer, J.: An improved incremental algorithm for computing prime implicants. In: Proc. AAAI-1992, San Jose, CA, pp. 780–785 (1992)
4. Jackson, P.: Computing prime implicants incrementally. In: Kapur, D. (ed.) CADE 1992. LNCS(LNAI), vol. 607, pp. 253–267. Springer, Heidelberg (1992)
5. Jackson, P., Pais, J.: Computing prime implicants. In: Stickel, M.E. (ed.) CADE 1990. LNCS(LNAI), vol. 449, pp. 543–557. Springer, Heidelberg (1990)
6. Kean, A., Tsiknis, G.: An incremental method for generating prime implicants/implicates. Journal of Symbolic Computation 9, 185–206 (1990)
7. Kean, A., Tsiknis, G.: Assumption based reasoning and clause management systems. Computational Intelligence 8(1), 1–24 (1992)
8. Manquinho, V.M., Flores, P.F., Silva, J.P.M., Oliveira, A.L.: Prime implicant computation using satisfiability algorithms. In: Proceedings of the IEEE International Conference on Tools with Artificial Intelligence, Newport Beach, USA, November 1997, pp. 232–239 (1997)
9. Morrison, D.R.: Patricia — practical algorithm to retrieve information coded in alphanumeric. Journal of the ACM 15(4), 514–534 (1968)
10. Murray, N.V., Rosenthal, E.: Efficient query processing with compiled knowledge bases. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS(LNAI), vol. 3702, pp. 231–244. Springer, Heidelberg (2005)
11. Murray, N.V., Rosenthal, E.: Efficient query processing with reduced implicate tries. Journal of Automated Reasoning 38(1-3), 155–172 (2007)
12. Murray, N.V., Rosenthal, E.: Updating reduced implicate tries. In: Olivetti, N. (ed.) TABLEAUX 2007. LNCS(LNAI), vol. 4548, pp. 183–198. Springer, Heidelberg (2007)
13. Ngair, T.: A new algorithm for incremental prime implicate generation. In: Proc. IJCAI-1993, Chambery, France (1993)
14. Przymusinski, T.C.: An algorithm to compute circumscription. Artificial Intelligence 38, 49–73 (1989)
15. Ramesh, A., Becker, G., Murray, N.V.: Cnf and dnf considered harmful for computing prime implicants/implicates. Journal of Automated Reasoning 18(3), 337–356 (1997)
16. Reiter, R., de Kleer, J.: Foundations of assumption-based truth maintenance systems: preliminary report. In: Proc. 6th National Conference on Artificial Intelligence, Seattle, WA, July 12-17, 1987, pp. 183–188 (1987)
17. Slagle, J.R., Chang, C.L., Lee, R.C.T.: A new algorithm for generating prime implicants. IEEE transactions on Computers C-19(4), 304–310 (1970)
18. Strzemecki, T.: Polynomial-time algorithm for generation of prime implicants. Journal of Complexity 8, 37–63 (1992)

Proof Systems for a Gödel Modal Logic

George Metcalfe[1],* and Nicola Olivetti[2]

[1] Department of Mathematics, Vanderbilt University
1326 Stevenson Center, Nashville TN 37240, USA
george.metcalfe@vanderbilt.edu

[2] LSIS-UMR CNRS 6168, Université Paul Cézanne
Campus de Saint Jérôme, Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20, France
nicola.olivetti@univ-cezanne.fr

Abstract. A basic propositional modal fuzzy logic $\mathsf{GK}_\Box$ is defined by combining the Kripke semantics of the modal logic K with the many-valued semantics of Gödel logic G. A sequent of relations calculus is introduced for $\mathsf{GK}_\Box$ and a constructive counter-model completeness proof is given. This calculus is used to establish completeness for a Hilbert-style axiomatization and Gentzen-style hypersequent calculus admitting cut-elimination, and to show that the logic is PSPACE-complete.

1 Introduction

Logical formalizations of vagueness and modal notions such as necessity, knowledge, and obligation have been studied intensively, the former primarily as *fuzzy logics* (see e.g. [14,16]), the latter under the rubric of *modal logics* (see e.g. [7]). However, there have been few treatments and until recently no systematic study of *modal fuzzy logics*. Such an investigation is important for providing a unified approach for topics such as fuzzy description logics, which can be understood, analogously to classical description logics, as multi-modal fuzzy logics [19,15].

Particular examples of fuzzy modal logics given in the literature have typically been situated quite far up the spectrum of modal logics, e.g. at the level of the logic $\mathsf{S5}$ (see e.g. [14]) or focus just on the minimal fuzzy logic of Zadeh (see e.g. Zhang [21]). More general approaches dealing with many-valued modal logics, such as [11,12], have focussed on the finite-valued case. Recent papers of Priest [17] and Bou et al. [5] provide a broad basis for studying fuzzy modal logics but again the majority of the results concern finite-valued modal logics. Deferring formal definitions to the next section, the rough idea of these approaches (followed also in this paper) is to consider Kripke models where the accessibility relation between worlds may be either Boolean-valued or many-valued. Propositional connectives operate as usual for the logic in question at an individual world, while the values of boxed formulas $\Box A$ are calculated using the infimum of values of A at accessible (to some degree) worlds. Validity is defined as usual as truth (i.e. taking the value 1) at all worlds of all models.

* Supported by a visiting professor grant at LSIS, Université Paul Cézanne, June 2008.

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 265–279, 2009.

In this paper, we narrow our focus on the fuzzy side to Gödel logic G, the infinite-valued version of a family of finite-valued logics introduced by Gödel in the 1930s [13], axiomatized by Dummett by adding the schema $(A \to B) \vee (B \to A)$ to intuitionistic logic [9]. Aside from being an important fuzzy logic, there is also a good practical reason to focus on G in modal contexts. As noted in [5], G is the only fuzzy logic whose modal analogues admit the schema $\Box(A \to B) \to (\Box A \to \Box B)$ (roughly speaking, since G, unlike other fuzzy logics, admits both weakening and contraction). Moreover, axiomatizations for a basic "K" Gödel modal logic with the $\Box$ or the $\Diamond$ modality (so far not both) have recently been provided by Caicedo and Rodríguez [6]. These authors also show, interestingly, that the logic with $\Box$ is complete with respect to either Boolean or many-valued accessibility relations but does not have the finite model property, while the logic with $\Diamond$ based on a many-valued accessibility relation has the finite model property but differs from the corresponding logic with a Boolean-valued relation.

We extend the work of [6] here by providing proof systems for the Gödel modal logic with $\Box$, the broader aim being to initiate a general investigation into the proof theory of modal fuzzy logics (e.g., as undertaken for fuzzy logics in [16]). A wide range of proof calculi have been developed for Gödel logic, including the sequent calculi of Sonobe [18] and Dyckhoff [10]; here we focus on extending the hypersequent calculus of Avron [2] and sequent of relations calculus of Baaz and Fermüller [4]. In particular, we give a constructive proof of completeness for the more proof-search-oriented sequent of relations calculus and use it to give both an alternative completeness proof for the axiomatization of Caicedo and Rodríguez [6] and a (first) proof of PSPACE-completeness. We then establish completeness and cut-elimination for the more elegant extended hypersequent calculus, better suited e.g. for extension to first-order modal fuzzy logics or investigating theoretical properties such as interpolation.

Note that the approach taken both here and in [6] for Gödel logic differs markedly from certain other developments in the literature. In particular, the intermediate logics extended with modalities investigated in e.g. [20] make use of two accessibility relations in Kripke models, one for the modal operator and another for the intuitionistic connectives. Also, the class of modalities for fuzzy logics considered in [8] represent truth stressers such as "very true" and, unlike the modalities considered here, can be interpreted as unary functions on $[0, 1]$.

2 The Gödel Modal Logic $\mathsf{GK}_\Box$

We make use of a language $\mathcal{L}_\Box$ with a countably infinite set Var of variables $p, q, r \ldots$, binary connectives $\to$, $\wedge$, $\vee$, constants $\bot$, $\top$, and a unary connective $\Box$. The set Fm of formulas, denoted $A, B, C \ldots$, is defined inductively as usual, and the complexity of a formula A, denoted $|A|$, is the number of connectives occurring in A. We also let $\neg A =_{\text{def}} A \to \bot$ and $A \leftrightarrow B =_{\text{def}} (A \to B) \wedge (B \to A)$. We use $\Gamma, \Pi, \Sigma, \Delta$ to stand for finite multisets of formulas, writing $\bigvee \Gamma$ and $\bigwedge \Gamma$ with $\bigvee [] =_{\text{def}} \bot$ and $\bigwedge [] =_{\text{def}} \top$ for disjunctions and conjunctions of formulas and $\Box \Gamma$ for $[\Box A : A \in \Gamma]$. We let $\Gamma^0 =_{\text{def}} []$ and $\Gamma^{n+1} =_{\text{def}} \Gamma \uplus \Gamma^n$ for $n \in \mathbb{N}$.

Following similar ideas in [14,6,5], the usual definition of the modal logic K can be generalized to define a natural "Gödel modal logic" $\mathsf{GK}_\Box$. Recall that a *(standard) Kripke frame* is a pair $\langle W, R\rangle$ where W is a non-empty set of *worlds* and $R \subseteq W^2$ is a binary *accessibility relation* on W. A *Kripke model for* $\mathsf{GK}_\Box$ is then a 3-tuple $K = \langle W, R, V\rangle$ where $\langle W, R\rangle$ is a Kripke frame and $V : \mathit{Var} \times W \to [0,1]$ is a mapping, called a *valuation*, extended to $V : \mathit{Fm} \times W \to [0,1]$ by:

- $V(A \to B, w) = \begin{cases} V(B,w) & \text{if } V(A,w) > V(B,w) \\ 1 & \text{otherwise;} \end{cases}$
- $V(A \wedge B, w) = \min(V(A,w), V(B,w))$;
- $V(A \vee B, w) = \max(V(A,w), V(B,w))$;
- $V(\top, w) = 1$ and $V(\bot, w) = 0$;
- $V(\Box A, w) = \inf(\{1\} \cup \{V(A,w') : Rww'\})$.

A formula A is $\mathsf{GK}_\Box$*-valid*, written $\models_{\mathsf{GK}_\Box} A$, if $V(A,w) = 1$ for all Kripke models $\langle W, R, V\rangle$ for $\mathsf{GK}_\Box$ and $w \in W$.

The above definitions give a reasonable semantics for a Gödel modal logic. However, we have made a number of significant "design choices". First, note that we have omitted the dual modality $\Diamond$, characterized by the condition:

- $V(\Diamond A, w) = \sup(\{0\} \cup \{V(A,w') : Rww'\})$.

This connective is definable using negation in (classical) modal logic as $\Diamond A =_{\text{def}} \neg\Box\neg A$, but for Gödel logic, the equivalence breaks down. We concentrate here on the fuzzy logic with just one modality simply as a first step in our investigations of the full logic GK.

A second complaint may be that $\mathsf{GK}_\Box$ is not "fuzzy enough" since the accessibility relation R of a Kripke frame is Boolean-valued (crisp). Consider instead a *fuzzy Kripke frame* as a pair $\langle W, R\rangle$ where W is a non-empty set of worlds and $R : W \times W \to [0,1]$ is a binary *fuzzy accessibility relation* on W. A *fuzzy Kripke model for* $\mathsf{GK}^{\mathsf{F}}_\Box$ is then a 3-tuple $K = \langle W, R, V\rangle$ where $\langle W, R\rangle$ is a fuzzy Kripke frame and $V : \mathit{Var} \times W \to [0,1]$ is a mapping defined as for $\mathsf{GK}_\Box$ except that the $\Box$ condition is changed (following the semantics of Gödel implication) to:

- $V(\Box A, w) = \inf(\{1\} \cup \{V(A,w') : Rww' > V(A,w')\})$.

A formula A is $\mathsf{GK}^{\mathsf{F}}_\Box$*-valid*, written $\models_{\mathsf{GK}^{\mathsf{F}}_\Box} A$, if $V(A,w) = 1$ for all fuzzy Kripke models $\langle W, R, V\rangle$ for $\mathsf{GK}^{\mathsf{F}}_\Box$ and $w \in W$. A dual treatment of modal many-valued logics based on classical and fuzzy Kripke models is a common theme in the literature (see e.g. [11,12,5,6]). However, for this paper the distinction is not so important. It has been shown in [6] (and will follow from our results below) that in the case of $\mathsf{GK}_\Box$ and $\mathsf{GK}^{\mathsf{F}}_\Box$ the valid formulas of the two logics coincide.

Another issue is the fact that in the definition of $\Box A$, the infimum value may not be "witnessed" by the value of A at any accessible world. If we restrict to worlds where this is always the case, i.e., by insisting that every infimum is a minimum, then we get a different "witnessed" logic (see e.g. [15]). For example, the formula $\Box\neg\neg p \to \neg\neg\Box p$ (considered in [6]) is valid in the witnessed logic,

$$\begin{array}{ll}
(A1)\ A \to (B \to A) & (A8)\ (A \to B) \to ((C \to A) \to (C \to B)) \\
(A2)\ (A \wedge B) \to A & (A9)\ (A \to (B \to C)) \to (B \to (A \to C)) \\
(A3)\ (A \wedge B) \to B & (A10)\ ((A \to C) \wedge (B \to C)) \to ((A \vee B) \to C) \\
(A4)\ A \to (B \to (A \wedge B)) & (A11)\ (A \to (B \to C)) \to ((A \wedge B) \to C) \\
(A5)\ \bot \to A & (A12)\ ((C \to A) \wedge (C \to B)) \to (C \to (A \wedge B)) \\
(A6)\ A \to (A \vee B) & (A13)\ (A \to (A \to B)) \to (A \to B) \\
(A7)\ B \to (A \vee B) & (A14)\ (A \to B) \vee (B \to A)
\end{array}$$

$$\frac{A \quad A \to B}{B}\ (\textsc{mp})$$

Fig. 1. The Hilbert System HG

but not in $\mathsf{GK}_\Box$: just consider a Kripke model $\langle \mathbb{N}, R, V\rangle$ where Rmn holds for all $m, n \in \mathbb{N}$ and $V(p, n) = 1/(n + 1)$ for all $n \in \mathbb{N}$. In fact, the formula is valid in all Kripke models with a finite number of worlds, so the logic $\mathsf{GK}_\Box$ does not have the finite model property.

An axiomatization of $\mathsf{GK}_\Box$ (and $\mathsf{GK}^\mathsf{F}_\Box$, since the valid formulas of these logics coincide) is obtained by extending a standard axiomatization of Gödel logic such as that presented in Fig. 1 with the usual K axiom schema and also a further schema reflecting the fact, already implicit in classical logic, that $V(\neg\neg A, w) = 0$ if $V(A, w) = 0$ and 1 otherwise. More precisely, let $\mathsf{HGK}_\Box$ be HG extended with:

$$\begin{array}{ll}
(K_\Box) & \Box(A \to B) \to (\Box A \to \Box B) \\
(Z_\Box) & \neg\neg\Box A \to \Box\neg\neg A
\end{array} \qquad \text{and} \qquad \frac{A}{\Box A}\ (\textsc{nec}).$$

Completeness for this axiomatization was established using a counter-model construction by Caicedo and Rodríguez in [6]:

Theorem 1 ([6]). $\models_{\mathsf{GK}_\Box} A$ *iff* $\models_{\mathsf{GK}^\mathsf{F}_\Box} A$ *iff* $\vdash_{\mathsf{HGK}_\Box} A$.

In what follows, we give an alternative proof of this theorem as a byproduct of a completeness proof for a sequent of relations calculus for $\mathsf{GK}_\Box$.

3 A Sequent of Relations Calculus

A *sequent of relations* $\mathcal{S}$ is a finite set of ordered triples:

$$A_1 \lhd_1 B_1 \mid \ldots \mid A_n \lhd_n B_n$$

where A_i and B_i are formulas and $\lhd_i \in \{<, \le\}$ for $i = 1 \ldots n$. We say that a sequent of relations $\mathcal{S}$ is $\mathsf{GK}_\Box$-*valid*, written $\models_{\mathsf{GK}_\Box} \mathcal{S}$, if for all Kripke models $\langle W, R, V\rangle$ and $w \in W$: $V(A, w) \lhd V(B, w)$ for some $(A \lhd B) \in \mathcal{S}$.

Note that $\lhd \in \{\le, <\}$ is used here both syntactically, as a symbol in sequents of relations, and semantically, for interpreting those sequents of relations. Let us call a sequent of relations *atomic* if it contains only propositional variables

Axioms:

$$\frac{}{\mathcal{S} \mid A \leq A}\ (\textsc{id}) \qquad \frac{}{\mathcal{S} \mid A \leq \top}\ (\leq\top) \qquad \frac{}{\mathcal{S} \mid \bot \leq A}\ (\bot\leq) \qquad \frac{}{\mathcal{S} \mid \bot < \top}\ (<)$$

Structural Rules:

$$\frac{\mathcal{S} \mid A \leq B \mid C \lhd D \mid A \leq D \quad \mathcal{S} \mid A \leq B \mid C \lhd D \mid C \leq B}{\mathcal{S} \mid A \leq B \mid C \lhd D}\ (\textsc{com}) \quad \frac{\mathcal{S} \mid \top \leq \bot}{\mathcal{S}}\ (\leq) \quad \frac{\mathcal{S}}{\mathcal{S} \mid A \lhd B}\ (\textsc{ew})$$

Logical Rules:

$$\frac{\mathcal{S} \mid A \lhd C \mid B \lhd C}{\mathcal{S} \mid A \wedge B \lhd C}\ (\wedge\lhd) \qquad \frac{\mathcal{S} \mid C \lhd A \quad \mathcal{S} \mid C \lhd B}{\mathcal{S} \mid C \lhd A \wedge B}\ (\lhd\wedge)$$

$$\frac{\mathcal{S} \mid A \lhd C \quad \mathcal{S} \mid B \lhd C}{\mathcal{S} \mid A \vee B \lhd C}\ (\vee\lhd) \qquad \frac{\mathcal{S} \mid C \lhd A \mid C \lhd B}{\mathcal{S} \mid C \lhd A \vee B}\ (\lhd\vee)$$

$$\frac{\mathcal{S} \mid B < A \quad \mathcal{S} \mid B < C}{\mathcal{S} \mid A \to B < C}\ (\to<) \qquad \frac{\mathcal{S} \mid A \leq B \mid C < B \quad \mathcal{S} \mid C < \top}{\mathcal{S} \mid C < A \to B}\ (<\to)$$

$$\frac{\mathcal{S} \mid \top \leq C \mid B < A \quad \mathcal{S} \mid B \leq C}{\mathcal{S} \mid A \to B \leq C}\ (\to\leq) \qquad \frac{\mathcal{S} \mid A \leq B \mid C \leq B}{\mathcal{S} \mid C \leq A \to B}\ (\leq\to)$$

Fig. 2. The Sequent of Relations Calculus SG

or constants, and *propositional* if it contains no occurrences of $\Box$. In Fig. 2, we present a sequent of relations calculus SG for G consisting of logical rules taken from [4] and some additional axioms and structural rules (based on similar calculi for G presented in [16]) for deriving valid atomic sequent of relations. As shown e.g. in [16], an atomic sequent of relations $\mathcal{S}$ is valid iff there exists $(a_i \lhd_i a_{i+1}) \in \mathcal{S}$ for $i = 1 \ldots n$ such that one of the following holds:

1. $a_1 = a_{n+1}$ or $a_1 = \bot$ or $a_{n+1} = \top$, where $\lhd_i$ is $\leq$ for some $i \in \{1, \ldots, n\}$.
2. $a_1 = \bot$ and $a_{n+1} = \top$.

It is an easy induction to show that an atomic sequent of relations $\mathcal{S}$ is valid iff it is derivable using the axioms and structural rules of SG (see e.g. [16] for very similar proofs). Since also the logical rules of SG are sound and invertible (see e.g. [16]), it follows that:

Theorem 2. *$\models_{\mathsf{GK}_\Box} \mathcal{S}$ iff $\vdash_{\mathsf{SG}} \mathcal{S}$ for any propositional sequent of relations $\mathcal{S}$.*

The sequent of relations calculus $\mathsf{SGK}_\Box$ consists of SG extended with the rules:

$$\frac{A_1 \leq B \mid \ldots \mid A_n \leq B \mid C_1 \leq \bot \mid \ldots \mid C_m \leq \bot}{\mathcal{S} \mid \Box A_1 \leq \Box B \mid \ldots \mid \Box A_n \leq \Box B \mid \Box C_1 \leq \bot \mid \ldots \mid \Box C_m \leq \bot}\ (\Box) \quad (n \geq 1, m \geq 0)$$

$$\frac{\mathcal{S} \mid A \leq B \mid \Box\top \leq B}{\mathcal{S} \mid A \leq B}\ (\textsc{wl}) \qquad \frac{\mathcal{S} \mid A \leq B \mid A \leq \bot}{\mathcal{S} \mid A \leq B}\ (\textsc{wr})$$

Note that the rather ugly "weakening" rules (WL) and (WR) allow us to work backwards from relations of the form $A \leq \Box B$ and $\Box A \leq B$ to $\Box\top \leq \Box B$ and $\Box A \leq \bot$, respectively, so that they fit the format of the rule $(\Box)$.

Example 1. Consider the following derivation of a sequent of relations corresponding to the K axiom schema $\Box(A \to B) \to (\Box A \to \Box B)$:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \leq B \mid B < A \mid A \leq A}\,\text{(ID)} \quad \dfrac{}{A \leq B \mid B < A \mid B \leq B}\,\text{(ID)}}{A \leq B \mid B < A}\,\text{(COM)}}{A \leq B \mid \top \leq B \mid B < A}\,\text{(EW)} \qquad \dfrac{}{A \leq B \mid B \leq B}\,\text{(ID)}}{A \leq B \mid A \to B \leq B}\,(\to\leq)}{\Box A \leq \Box B \mid \Box(A \to B) \leq \Box B}\,(\Box)}{\Box(A \to B) \leq \Box A \to \Box B}\,(\leq\to)}{\Box(A \to B) \leq \Box A \to \Box B \mid \top \leq \Box A \to \Box B}\,\text{(EW)}}{\top \leq \Box(A \to B) \to (\Box A \to \Box B)}\,(\leq\to)
$$

Theorem 3. *If* $\vdash_{\mathsf{SGK}_\Box} \mathcal{S}$*, then* $\models_{\mathsf{GK}_\Box} \mathcal{S}$*.*

Proof. The proof is a standard induction on the height of a derivation of $\mathcal{S}$ in $\mathsf{SGK}_\Box$; the only significantly new case is to establish the soundness of $(\Box)$. Suppose that there is a counter-model $K = \langle W, R, V \rangle$ for the conclusion. I.e. for some $w \in W$: $V(\Box A_i, w) > V(\Box B, w)$ for $i = 1 \ldots n$ and $V(\Box C_j, w) > 0$ for $j = 1 \ldots m$. Then there must be a world w' accessible to w where $V(A_i, w') > V(B, w')$ for $i = 1 \ldots n$. Moreover, since $V(\Box C_j, w) \neq 0$, we must also have $V(C_j, w') > 0$ for $j = 1 \ldots m$ as required. □

Completeness is of course more complicated. Notice first that all the logical rules (as proved in [4]), (COM), (WL), and (WR) are invertible in the sense that if the conclusion is $\mathsf{GK}_\Box$-valid, then so are the premises. Hence applying the logical rules backwards to a $\mathsf{GK}_\Box$-valid sequent of relations results in $\mathsf{GK}_\Box$-valid sequent of relations containing only modal formulas and atoms. Let us call a sequent of relations $\mathcal{S}$ containing only modal formulas and atoms *saturated* if whenever $\mathcal{S}$ occurs as the conclusion of (COM), $(\leq)$, (WL), or (WR), then $\mathcal{S}$ also occurs as one of the premises. In other words, the sequent of relations is "closed" under applications of these rules. Moreover, since sequents of relations are sets of pairs of formulas, there is a finite number that can be obtained by applying the rules backwards to any given sequent of relations. Hence easily:

Lemma 1. *Every sequent of relations* $\mathcal{S}$ *is derivable from a set of saturated sequent of relations* $\mathcal{S}_1, \ldots, \mathcal{S}_n$ *using the logical rules,* (COM)*,* $(\leq)$*,* (WL)*, and* (WR)*, and if* $\models_{\mathsf{GK}_\Box} \mathcal{S}$*, then* $\models_{\mathsf{GK}_\Box} \mathcal{S}_i$ *for* $i = 1 \ldots n$*.*

The challenge now is to show that a valid saturated sequent of relations $\mathcal{S}$ is derivable. Our strategy will be to use Lemmas 2 and 4 to show that *either* $\mathcal{S}$ is derivable using just the rules of SG *or* the part of $\mathcal{S}$ containing only relations of the form $\Box A \leq \Box B$ and $\Box C \leq \bot$ is itself valid. For the latter case, $\mathcal{S}$ is derivable using $(\Box)$ from a less complex sequent of relations that is proved valid in Lemma 5. An inductive argument then completes the proof.

Let us say that a formula *occurs* in a sequent of relations if it occurs as the left or right side of one of the relations. Then a sequent of relations is said to be *propositionally* $\mathsf{GK}_\Box$*-valid* if the sequent of relations obtained by replacing each occurrence of a modal formula $\Box A$ with a variable p_A is $\mathsf{GK}_\Box$-valid.

Lemma 2. *Let $\mathcal{S} \mid \mathcal{S}'$ be saturated where $\mathcal{S}$ contains only relations of the form $\Box A \lhd \Box B$, $\Box A \leq \bot$, $\Box A < \top$, and $\bot < \Box B$, and $\mathcal{S}'$ contains no relations of this form. If $\models_{\mathsf{GK}_\Box} \mathcal{S} \mid \mathcal{S}'$, then either $\mathcal{S} \mid \mathcal{S}'$ is propositionally $\mathsf{GK}_\Box$-valid or $\models_{\mathsf{GK}_\Box} \mathcal{S}$.*

Proof. Proceeding contrapositively, suppose that $\not\models_{\mathsf{GK}_\Box} \mathcal{S}$ and $\mathcal{S} \mid \mathcal{S}'$ is *not* propositionally $\mathsf{GK}_\Box$-valid. Then for some model $K = \langle W, R, V\rangle$ and $w \in W$:

(1) $V(\Box A, w) \not\lhd V(\Box B, w)$ for all $(\Box A \lhd \Box B) \in \mathcal{S}$.
(2) $V(\Box A, w) > 0$ for all $(\Box A \leq \bot) \in \mathcal{S}$.
(3) $V(\Box A, w) = 1$ for all $(\Box A < \top) \in \mathcal{S}$.
(4) $V(\Box B, w) = 0$ for all $(\bot < \Box B) \in \mathcal{S}$.

For each $\Box A$ occurring in $\mathcal{S} \mid \mathcal{S}'$, let us add a constant c_A to the language so that for any model $\langle W', R', V'\rangle$ and world $x \in W'$:

$$V'(c_A, x) = V(\Box A, w).$$

Let $(\mathcal{S} \mid \mathcal{S}')^P$ be $\mathcal{S} \mid \mathcal{S}'$ with each $\Box A$ occurring in $\mathcal{S} \mid \mathcal{S}'$ replaced by c_A.

Claim: $\not\models_{\mathsf{GK}_\Box} (\mathcal{S} \mid \mathcal{S}')^P$.

The result follows from this claim. Let $v : Var \to [0,1]$ be the propositional counter-valuation for $(\mathcal{S} \mid \mathcal{S}')^P$. Define $K' = \langle W \cup \{w_0\}, R', V'\rangle$ where:

1. $R' = R \cup \{(w_0, w') : (w, w') \in R\}$.
2. V' is V extended with $V'(p, w_0) = v(p)$ for all variables p.

Then $V'(\Box A, w_0) = V(\Box A, w)$ for all $\Box A$ occurring in $\mathcal{S} \mid \mathcal{S}'$. So, since v is a counter-valuation for $(\mathcal{S} \mid \mathcal{S}')^P$, we have $\not\models_{\mathsf{GK}_\Box} \mathcal{S} \mid \mathcal{S}'$ as required.

Proof of claim. Proceeding by contraposition, suppose that $\models_{\mathsf{GK}_\Box} (\mathcal{S} \mid \mathcal{S}')^P$. Then using (1)-(4), $\models_{\mathsf{GK}_\Box} (\mathcal{S}')^P$. By saturation, there are several possibilities:

(i) $(\mathcal{S}')^P$ contains $a \leq a$ or $\bot \leq a$ or $a \leq \top$ or $\bot < \top$. But then $\mathcal{S}'$ is propositionally $\mathsf{GK}_\Box$-valid, a contradiction.
(ii) $(\mathcal{S}')^P$ contains $c_C \lhd c_D$ or $c_C < \top$ or $\bot < c_D$. But then $\mathcal{S}'$ contains $\Box C \lhd \Box D$ or $\Box C < \top$ or $\bot < \Box D$, a contradiction.
(iii) $(\mathcal{S}')^P$ contains $a \leq c_D$ and $V(\Box D, w) = 1$. But then by (WL), $\mathcal{S}$ contains $\Box\top \leq \Box D$ and $V(\Box D, w) < 1$, a contradiction.
(iv) $(\mathcal{S}')^P$ contains $c_C \leq a$ and $V(\Box C, w) = 0$. But then by (WR), $\mathcal{S}$ contains $\Box C \leq \bot$ and $V(\Box C, w) > 0$, a contradiction. □

The next step is to remove relations involving strict inequalities. We make use of the following scaling lemmas, easily proved by induction on formula complexity.

Lemma 3. *Let $K = \langle W, R, V\rangle$ be a model and $c \in (0,1)$.*

(a) Let $K' = \langle W, R, V'\rangle$ where for each variable p and $x \in W$:

$$V'(p,x) = \begin{cases} 1 & \text{if } V(p,x) \geq c \\ V(p,x) & \text{otherwise.} \end{cases}$$

Then for all $x \in W$ and every formula A:

$$V'(A,x) = \begin{cases} 1 & \textit{if } V(A,x) \geq c \\ V(A,x) & \textit{otherwise.} \end{cases}$$

(b) Let $K' = \langle W, R, V' \rangle$ where $0 < a < b < 1$ and for each variable p and $x \in W$:

$$V'(p,x) = \begin{cases} a + c(V(p,x) - a) & \textit{if } V(p,x) \in (a,b] \\ V(p,x) & \textit{otherwise.} \end{cases}$$

Then for all $x \in W$ and every formula A:

$$V'(A,x) = \begin{cases} a + c(V(A,x) - a) & \textit{if } V(A,x) \in (a,b] \\ V(p,x) & \textit{otherwise.} \end{cases}$$

Lemma 4. *Let $\mathcal{S} \mid F < G$ be saturated and contain only relations of the form $\Box A \lhd \Box B$, $\Box A \leq \bot$, $\Box A < \top$, and $\bot < \Box B$. If $\models_{\mathsf{GK}_\Box} \mathcal{S} \mid F < G$, then $\models_{\mathsf{GK}_\Box} \mathcal{S}$.*

Proof. Let us consider just the case where $F < G$ is of the form $\Box A < \Box B$, since the other cases are very similar. Proceeding contrapositively, suppose that $\not\models_{\mathsf{GK}_\Box} \mathcal{S}$. Hence there is a model $K = \langle W, R, V \rangle$ and $w \in W$ such that:

(1) $V(\Box C, w) \not\lhd V(\Box D, w)$ for all $(\Box C \lhd \Box D) \in \mathcal{S}$.
(2) $V(\Box C, w) > 0$ for all $(\Box C \leq \bot) \in \mathcal{S}$.
(3) $V(\Box C, w) = 1$ for all $(\Box C < \top) \in \mathcal{S}$.
(4) $V(\Box D, w) = 0$ for all $(\bot < \Box D) \in \mathcal{S}$.

If $V(\Box A, w) \geq V(\Box B, w)$, then $\not\models_{\mathsf{GK}_\Box} \mathcal{S} \mid \Box A < \Box B$ as required, so assume that:

(5) $V(\Box A, w) < V(\Box B, w)$.

Since $\mathcal{S} \mid F < G$ is saturated, for each $(\Box C \leq \Box D) \in \mathcal{S}$:

either $(\Box C \leq \Box B) \in \mathcal{S}$ and, by (1), $V(\Box C, w) > V(\Box B, w)$
or $(\Box A \leq \Box D) \in \mathcal{S}$ and, by (1), $V(\Box A, w) > V(\Box D, w)$

and in particular:

(6) $V(\Box A, w) \leq V(\Box D, w) < V(\Box C, w) \leq V(\Box B, w)$ is not possible.

We have two cases:

(i) Suppose that $V(\Box B, w) < 1$. Then using Lemma 3 (b), we define for each $i \in \mathbb{Z}^+$, a counter-model $K_i = \langle W_i, R_i, V_i \rangle$ where:
 1. $\langle W_i, R_i \rangle$ is a copy of $\langle W, R \rangle$ with distinct worlds for each $i \in \mathbb{Z}^+$ and w_i is the corresponding copy of w.
 2. For all formulas E satisfying $V_i(\Box A, w_i) < V_i(E, w_i) \leq V_i(\Box B, w_i)$: $V_i(\Box A, w_i) < V_i(E, w_i) < V(\Box A, w_i) + 1/i$.

 Now we define a model $K' = \langle W', R', V' \rangle$ where:

1. $W' = \{w_0\} \cup \bigcup_{i \in \mathbb{Z}^+} W_i$.
2. $R' = \{(w_0, w') : (w_i, w') \in V_i$ for some $i \in \mathbb{Z}^+\} \cup \bigcup_{i \in \mathbb{Z}^+} R_i$.
3. $V'(p, x) = V_i(p, x)$ for all $x \in W_i$ and $V'(p, w_0) = 0$.

But then: $$\begin{aligned} V'(\Box B, w_0) &= \inf(\{1\} \cup \{V'(B, w') : R' w_0 w'\}) \\ &= \inf\{V_i(\Box B, w_i) : i \in \mathbb{Z}^+\} \\ &= V'(\Box A, w_0). \end{aligned}$$

If $V(C, w) \geq V(D, w)$ for some $\Box C < \Box D \in \mathcal{S}$, then $V'(\Box C, w_0) \geq V'(\Box D, w_0)$. If $\Box C \leq \Box D \in \mathcal{S}$, then by (6), it follows that $V'(\Box C, w_0) > V'(\Box D, w_0)$.

(ii) Now suppose that $V(\Box B, w) = 1$. Then using Lemma 3 (a), we obtain a model $K' = \langle W, R, V' \rangle$ where $V'(\Box A, w) = V'(\Box B, w) = 1$. As in case (i), again by (6), the other inequalities required are not affected. □

Lemma 5. *Suppose that* $\models_{\mathsf{GK}_\Box} \{\Box A_i \leq \Box B_i\}_{i=1}^n \mid \{\Box C_j \leq \bot\}_{j=1}^m$. *Then* $\models_{\mathsf{GK}_\Box} \bigwedge_{i \in I} A_i \leq \bigwedge_{i \in I} B_i \mid \{C_j \leq \bot\}_{j=1}^m$ *for some* $\emptyset \subset I \subseteq \{1, \ldots, n\}$.

Proof. We argue by contraposition; i.e., suppose that:

$$\not\models_{\mathsf{GK}_\Box} \bigwedge_{i \in I} A_i \leq \bigwedge_{i \in I} B_i \mid \{C_j \leq \bot\}_{j=1}^m \quad \text{for all} \quad \emptyset \subset I \subseteq \{1, \ldots, n\}.$$

We obtain a model for each $i \in \{1, \ldots, n\}$ as follows. By assumption:

$$\not\models_{\mathsf{GK}_\Box} A_i \wedge \ldots \wedge A_n \leq B_i \wedge \ldots \wedge B_n \mid \{C_j \leq \bot\}_{j=1}^m.$$

So we have $K_i = \langle W_i, R_i, V_i \rangle$ and $x_i \in W_i$ (with each W_i distinct) such that:

$$V_i(A_i \wedge \ldots \wedge A_n, x_i) > V_i(B_i \wedge \ldots \wedge B_n, x_i) \quad \text{and} \quad V_i(C_j) > 0 \quad \text{for } j = 1 \ldots m.$$

Moreover, without loss of generality we can assume:

$$V_i(B_i, x_i) \leq V_i(B_k, x_i) \quad \text{and so} \quad V_i(A_k, x_i) > V_i(B_i, x_i) \quad \text{for } k = i \ldots n.$$

Now using Lemma 3, we define iteratively $K_i' = \langle W_i, R_i, V_i' \rangle$ for $i = n \ldots 1$ such that for $j = i \ldots n$:

(1) $V_j'(B_j, x_j) < V_k(A_j, x_k)$ for $k = 1 \ldots i - 1$.
(2) $V_j'(B_j, x_j) < V_k'(A_j, x_k)$ for $k = i \ldots n$.
(3) $V_j'(C_k, x_j) > 0$ for $k = 1 \ldots m$.

We achieve this by scaling the interval $[0, V_i(B_i, x_i)]$ (if necessary) to the smaller interval $[0, V_i'(B_i, x_i)]$ so that (1) and (2) hold with j replaced by i. Now consider $j \in \{i, \ldots, n\}$. Clearly $V_i'(B_j, x_j) \leq V_i(B_j, x_j)$. Moreover, since $V_i(A_j, x_i) > V_i(B_i, x_i)$, also $V_i'(A_j, x_i) = V_i(A_j, x_i) > V_j'(B_j, x_j)$ as required.

Finally, we define a model $K = \langle W, R, V \rangle$ where for a new world w_0:

1. $W = W_1 \cup \ldots \cup W_n \cup \{w_0\}$.
2. $R = R_1 \cup \ldots \cup R_n \cup \{(w_0, w) : (x_i, w) \in R_i$ for some $i \in \{1, \ldots, n\}\}$.
3. $V(p, x) = V_i'(p, x)$ for all $x \in W_i$ and $V(p, w_0) = 0$.

Then by (1)-(3) above, $V(\Box B_i, x_0) = V(B_i, x_i) < V(\Box A_i, x_0)$ for $i = 1 \ldots n$ and $V(\Box C_k, x_0) > 0$ for $k = 1 \ldots m$ as required. □

Theorem 4 (Completeness). *If* $\models_{\mathsf{GK}_\Box} \mathcal{S}$*, then* $\vdash_{\mathsf{SGK}_\Box} \mathcal{S}$.

Proof. We prove the theorem by induction on the modal degree of the sequent of relations $\mathcal{S}$: the maximal complexity of a boxed subformula occurring in $\mathcal{S}$. If the modal degree is 0, then $\mathcal{S}$ is propositional and $\mathsf{SGK}_\Box$-derivable. Otherwise, by Lemma 1 (since working upwards, the rules do not increase modal degree), we can assume that $\mathcal{S}$ is both $\mathsf{GK}_\Box$-valid and saturated. If $\mathcal{S}$ is propositionally $\mathsf{GK}_\Box$-valid, then it is derivable. Otherwise, by Lemmas 2 and 4, $\mathcal{S}$ is of the form:

$$\mathcal{S}' \mid \{\Box A_i \leq \Box B_i\}_{i=1}^{n} \mid \{\Box C_j \leq \bot\}_{j=1}^{m}$$

and $\models_{\mathsf{GK}_\Box} \{\Box A_i \leq \Box B_i\}_{i=1}^{n} \mid \{\Box C_j \leq \bot\}_{j=1}^{m}$. But then by Lemma 5:

$$\models_{\mathsf{GK}_\Box} \bigwedge_{i \in I} A_i \leq \bigwedge_{i \in I} B_i \mid \{C_j \leq \bot\}_{j=1}^{m} \quad \text{for some } \emptyset \subset I \subseteq \{1, \ldots, n\}.$$

Let us assume without loss of generality that $I = \{1, \ldots, n\}$. Then also:

$$\models_{\mathsf{GK}_\Box} \{A_i \leq B_k\}_{i=1}^{n} \mid \{C_j \leq \bot\}_{j=1}^{m} \quad \text{for } k = 1 \ldots n.$$

So by the induction hypothesis and an application of (□):

$$\vdash_{\mathsf{SGK}} \mathcal{S}' \mid \{\Box A_i \leq \Box B_k\}_{i=1}^{n} \mid \{\Box C_j \leq \bot\}_{j=1}^{m} \quad \text{for } k = 1 \ldots n.$$

But then $\mathcal{S}$ is derivable by repeated applications of (COM). □

We can use $\mathsf{SGK}_\Box$ to give an alternative completeness proof for the axiomatization $\mathsf{HGK}_\Box$ with respect to both standard and fuzzy Kripke frames (Theorem 1).

Corollary 1. $\models_{\mathsf{GK}_\Box} A$ *iff* $\models_{\mathsf{GK}^{\mathsf{F}}_\Box} A$ *iff* $\vdash_{\mathsf{HGK}_\Box} A$.

Proof. First notice that the soundness proof for $\mathsf{SGK}_\Box$ can be adjusted to prove soundness for fuzzy frames: i.e., if $\vdash_{\mathsf{SGK}_\Box} \mathcal{S}$, then $\models_{\mathsf{GK}^{\mathsf{F}}_\Box} \mathcal{S}$. But then it follows from the completeness proof (the other direction is trivial) that $\models_{\mathsf{GK}^{\mathsf{F}}_\Box} \mathcal{S}$ iff $\models_{\mathsf{GK}_\Box} \mathcal{S}$. Now consider the following interpretation of sequent of relations:

$$I(\{A_i < B_i\}_{i=1}^{n} \mid \{C_j \leq D_j\}_{j=1}^{m}) = \bigwedge_{i=1}^{n}(B_i \to A_i) \to \bigvee_{j=1}^{m}(C_j \to D_j).$$

It is easily shown that $\models_{\mathsf{GK}_\Box} \mathcal{S}$ iff $\models_{\mathsf{GK}_\Box} I(\mathcal{S})$. Hence to establish completeness for $\mathsf{HGK}_\Box$, it is sufficient to show that for each rule $\mathcal{S}_1, \ldots, \mathcal{S}_n \ / \ \mathcal{S}$ of $\mathsf{SGK}_\Box$, whenever $\vdash_{\mathsf{HGK}_\Box} I(\mathcal{S}_i)$ for $i = 1 \ldots n$, also $\vdash_{\mathsf{HGK}_\Box} I(\mathcal{S})$. This is straightforward for the propositional rules and easy for the weakening rules, so let us just consider □. We show the simplified case that if $\vdash_{\mathsf{HGK}_\Box} (A \to B) \vee \neg C$, then $\vdash_{\mathsf{HGK}_\Box} (\Box A \to \Box B) \vee \neg \Box C$. Note first that $\vdash_{\mathsf{HG}} (\neg\neg F \to G) \leftrightarrow (G \vee \neg F)$. Hence if $\vdash_{\mathsf{HGK}_\Box} (A \to B) \vee \neg C$, then $\vdash_{\mathsf{HGK}_\Box} \neg\neg C \to (A \to B)$. But then by (NEC), $\vdash_{\mathsf{HGK}_\Box} \Box(\neg\neg C \to (A \to B))$ and using $(K_\Box)$, $\vdash_{\mathsf{HGK}_\Box} \Box\neg\neg C \to (\Box A \to \Box B)$. Now, using $(Z_\Box)$, $\vdash_{\mathsf{HGK}_\Box} \neg\neg\Box C \to (\Box A \to \Box B)$. But then $\vdash_{\mathsf{HGK}_\Box} (\Box A \to \Box B) \vee \neg\Box C$ as required. □

Theorem 5. *The* $\mathsf{GK}_\Box$*-validity problem for* $\mathsf{GK}_\Box$ *is PSPACE-complete.*

Proof. First we show that $\mathsf{GK}_\Box$ is PSPACE-hard. We recall that the modal logic K is PSPACE-complete (see e.g. [7]). Consider the translation $*$ sending each propositional variable p to its double negation $\neg\neg p$. We can easily show that $\models_{\mathsf{K}} A$ iff $\models_{\mathsf{GK}_\Box} A^*$ which establishes that $\mathsf{GK}_\Box$ must also be PSPACE-hard. For the non-trivial direction consider any GK model $M = \langle W, R, V\rangle$ and define a K model $M' = \langle W, R, V'\rangle$ by stipulating: $V'(p, w) = V(\neg\neg p, w)$. Then by a simple induction, $V'(C, w) = V(C^*, w) \in \{0, 1\}$ for any formula C. Thus if $\not\models_{\mathsf{GK}_\Box} A^*$, then there is a K model M where A is not valid.

For PSPACE-inclusion, we consider the sequent of relations calculus $\mathsf{SGK}_\Box$. Given a formula A, let $Sub(A)$ be the set of subformulas of A together with the formulas $\top$, $\Box\top, \bot, \Box\bot$, and consider the set

$$\Phi_A = \{C \lhd D : C, D \in Sub(A), \lhd \in \{<, \leq\}\}.$$

The cardinality of Φ_A is $O(|A|^2)$. Since any sequent of relations appearing in a derivation of $\top \leq A$ is a subset of Φ_A, its size is also $O(|A|^2)$.

We now consider the length of branches in the search for an $\mathsf{SGK}_\Box$-derivation of $\top \leq A$. Using the invertibility of the logical rules we assume that any branch of a derivation is expanded by applying iteratively the rules upwards in the following order:

(1) Apply the logical rules, (WL), (WR), ($\leq$), and (COM) in order to obtain a saturated sequent and check the axioms.
(2) Apply ($\Box$) and restart from (1) with the premise of this rule.

The length of the branch built in (1) is $O(|A|^2)$ since each logical rule replaces one relation with one or two relations involving formulas of smaller complexity, and each application of (WL), (WR), ($\leq$), and (COM) add exactly one relation at a time, with the total number of different relations possible being $O(|A|^2)$. The sequent obtained in (2) by applying ($\Box$) has a smaller or equal number of relations and a strictly smaller modal degree. The entire length of a proof branch is hence bounded by $O(|A|^2 \times m) = O(|A|^3)$, where m is the modal degree of A.

Thus storing a branch of a proof requires only polynomial space. Moreover, the branching is at most binary. As usual, we search for a proof in a depth-first manner: we store one branch at a time together with some information (requiring a small amount of space, say logarithmic space) to reconstruct branching points and backtracking points, the latter determined by alternative applications of ($\Box$). Hence the total amount of space needed for carrying out proof search is polynomial in the size of A, and so deciding validity for $\mathsf{GK}_\Box$ is in PSPACE. □

4 A Hypersequent Calculus

Hypersequents were introduced by Avron in [1] as a generalization of Gentzen sequents that allow disjunctive or parallel forms of reasoning. Instead of a single sequent, there is a collection of sequents that can be "worked on" simultaneously.

Initial Hypersequents

$$\frac{}{\mathcal{G} \mid A \Rightarrow A}\ (\textsc{id}) \qquad \frac{}{\mathcal{G} \mid \Gamma, \bot \Rightarrow \Delta}\ (\bot\Rightarrow) \qquad \frac{}{\mathcal{G} \mid \Gamma \Rightarrow \top}\ (\Rightarrow\top)$$

Structural Rules

$$\frac{\mathcal{G}}{\mathcal{G} \mid \mathcal{H}}\ (\textsc{ew}) \qquad \frac{\mathcal{G} \mid \mathcal{H} \mid \mathcal{H}}{\mathcal{G} \mid \mathcal{H}}\ (\textsc{ec}) \qquad \frac{\mathcal{G} \mid \Gamma_1, \Pi_1 \Rightarrow \Delta_1 \quad \mathcal{G} \mid \Gamma_2, \Pi_2 \Rightarrow \Delta_2}{\mathcal{G} \mid \Gamma_1, \Gamma_2 \Rightarrow \Delta_1 \mid \Pi_1, \Pi_2 \Rightarrow \Delta_2}\ (\textsc{com})$$

$$\frac{\mathcal{G} \mid \Gamma \Rightarrow \Delta}{\mathcal{G} \mid \Gamma, A \Rightarrow \Delta}\ (\textsc{wl}) \qquad \frac{\mathcal{G} \mid \Gamma \Rightarrow}{\mathcal{G} \mid \Gamma \Rightarrow A}\ (\textsc{wr}) \qquad \frac{\mathcal{G} \mid \Gamma, A, A \Rightarrow \Delta}{\mathcal{G} \mid \Gamma, A \Rightarrow \Delta}\ (\textsc{cl})$$

Logical Rules

$$\frac{\mathcal{G} \mid \Gamma_1 \Rightarrow A \quad \mathcal{G} \mid \Gamma_2, B \Rightarrow \Delta}{\mathcal{G} \mid \Gamma_1, \Gamma_2, A \to B \Rightarrow \Delta}\ (\to\Rightarrow) \qquad \frac{\mathcal{G} \mid \Gamma, A \Rightarrow B}{\mathcal{G} \mid \Gamma \Rightarrow A \to B}\ (\Rightarrow\to)$$

$$\frac{\mathcal{G} \mid \Gamma, A \Rightarrow \Delta}{\mathcal{G} \mid \Gamma, A \wedge B \Rightarrow \Delta}\ (\wedge\Rightarrow)_1 \qquad \frac{\mathcal{G} \mid \Gamma, B \Rightarrow \Delta}{\mathcal{G} \mid \Gamma, A \wedge B \Rightarrow \Delta}\ (\wedge\Rightarrow)_2 \qquad \frac{\mathcal{G} \mid \Gamma \Rightarrow A \quad \mathcal{G} \mid \Gamma \Rightarrow B}{\mathcal{G} \mid \Gamma \Rightarrow A \wedge B}\ (\Rightarrow\wedge)$$

$$\frac{\mathcal{G} \mid \Gamma, A \Rightarrow \Delta \quad \mathcal{G} \mid \Gamma, B \Rightarrow \Delta}{\mathcal{G} \mid \Gamma, A \vee B \Rightarrow \Delta}\ (\vee\Rightarrow) \qquad \frac{\mathcal{G} \mid \Gamma \Rightarrow A}{\mathcal{G} \mid \Gamma \Rightarrow A \vee B}\ (\Rightarrow\vee)_1 \qquad \frac{\mathcal{G} \mid \Gamma \Rightarrow B}{\mathcal{G} \mid \Gamma \Rightarrow A \vee B}\ (\Rightarrow\vee)_2$$

Cut Rule

$$\frac{\mathcal{G} \mid \Gamma_1, A \Rightarrow \Delta \quad \mathcal{G} \mid \Gamma_2 \Rightarrow A}{\mathcal{G} \mid \Gamma_1, \Gamma_2 \Rightarrow \Delta}\ (\textsc{cut})$$

Fig. 3. The Gentzen System GG

More precisely, we define a *sequent* S here as an ordered pair consisting of a finite multiset Γ of formulas and a multiset Δ containing at most one formula, written $\Gamma \Rightarrow \Delta$, and a *hypersequent* $\mathcal{G}$ as a non-empty finite multiset of sequents, written $\Gamma_1 \Rightarrow \Delta_1 \mid \ldots \mid \Gamma_n \Rightarrow \Delta_n$ or sometimes, for short, as $[\Gamma_i \Rightarrow \Delta_i]_{i=1}^n$. By taking multisets of formulas and sequents rather than sequences (as used e.g. by Avron in [1]) or sets we ensure that the multiplicity but not the order of elements is important. We interpret sequents and hypersequents as follows (recalling that $\bigwedge[] =_{\text{def}} \top$ and $\bigvee[] =_{\text{def}} \bot$):

$$\mathrm{I}(\Gamma \Rightarrow \Delta) =_{\text{def}} \bigwedge \Gamma \to \bigvee \Delta \qquad \mathrm{I}(S_1 \mid \ldots \mid S_n) =_{\text{def}} \mathrm{I}(S_1) \vee \ldots \vee \mathrm{I}(S_n).$$

Hypersequent calculi admitting cut-elimination have been defined for a wide range of fuzzy logics (for details see [16]). In particular, the first example of such a system was the calculus GG defined for Gödel logic by Avron in [2] (see also [3,16]). We extend this system, presented in Figure 3 with an adapted version of the usual Gentzen rule for K. $\mathsf{GGK}_\Box$ is GG extended with:

$$\frac{\Pi \Rightarrow \mid \Gamma \Rightarrow A}{\Box\Pi \Rightarrow \mid \Box\Gamma \Rightarrow \Box A}\ (\Box)$$

Rules for the defined negation $\neg A =_{\text{def}} A \to \bot$ are derivable in $\mathsf{GGK}_\Box$:

$$\frac{\mathcal{G} \mid \Gamma \Rightarrow A}{\mathcal{G} \mid \Gamma, \neg A \Rightarrow}\ (\neg\Rightarrow) \qquad \frac{\mathcal{G} \mid \Gamma, A \Rightarrow}{\mathcal{G} \mid \Gamma \Rightarrow \neg A}\ (\Rightarrow\neg)$$

It will also be helpful (e.g. in proving cut-elimination) to consider a generalization of the rule ($\Box$), derivable using (COM), ($\Box$), (CL), and (WL):

$$\frac{\varPi_1 \Rightarrow \mid \ldots \mid \varPi_n \Rightarrow \mid \varGamma \Rightarrow A}{\Box\varPi_1 \Rightarrow \mid \ldots \mid \Box\varPi_n \Rightarrow \mid \Box\varGamma \Rightarrow \Box A}\ (\Box)^* \qquad (n \in \mathbb{N})$$

Example 2. All the axioms of $\mathsf{HGK}_\Box$ are derivable in $\mathsf{GGK}_\Box$; e.g. for $(Z_\Box)$:

$$\begin{prooftree}
\AxiomC{}
\RightLabel{(ID)}
\UnaryInfC{$A \Rightarrow A$}
\RightLabel{$(\neg\Rightarrow)$}
\UnaryInfC{$A, \neg A \Rightarrow$}
\AxiomC{}
\RightLabel{(ID)}
\UnaryInfC{$A \Rightarrow A$}
\RightLabel{$(\neg\Rightarrow)$}
\UnaryInfC{$A, \neg A \Rightarrow$}
\RightLabel{(COM)}
\BinaryInfC{$A, A \Rightarrow \mid \neg A, \neg A \Rightarrow$}
\RightLabel{(CL)}
\UnaryInfC{$A, A \Rightarrow \mid \neg A \Rightarrow$}
\RightLabel{(CL)}
\UnaryInfC{$A \Rightarrow \mid \neg A \Rightarrow$}
\RightLabel{$(\Rightarrow\neg)$}
\UnaryInfC{$A \Rightarrow \mid \Rightarrow \neg\neg A$}
\RightLabel{$(\Box)$}
\UnaryInfC{$\Box A \Rightarrow \mid \Rightarrow \Box\neg\neg A$}
\RightLabel{$(\Rightarrow\neg)$}
\UnaryInfC{$\Rightarrow \neg\Box A \mid \Rightarrow \Box\neg\neg A$}
\RightLabel{$(\neg\Rightarrow)$}
\UnaryInfC{$\neg\neg\Box A \Rightarrow \mid \Rightarrow \Box\neg\neg A$}
\RightLabel{(WL)}
\UnaryInfC{$\neg\neg\Box A \Rightarrow \mid \neg\neg\Box A \Rightarrow \Box\neg\neg A$}
\RightLabel{(WR)}
\UnaryInfC{$\neg\neg\Box A \Rightarrow \Box\neg\neg A \mid \neg\neg\Box A \Rightarrow \Box\neg\neg A$}
\RightLabel{(EC)}
\UnaryInfC{$\neg\neg\Box A \Rightarrow \Box\neg\neg A$}
\RightLabel{$(\rightarrow\Rightarrow)$}
\UnaryInfC{$\Rightarrow \neg\neg\Box A \rightarrow \Box\neg\neg A$}
\end{prooftree}$$

Theorem 6. $\vdash_{\mathsf{GGK}_\Box} \mathcal{G}$ *iff* $\vdash_{\mathsf{HGK}_\Box} \mathrm{I}(\mathcal{G})$ *iff* $\models_{\mathsf{GK}_\Box} \mathrm{I}(\mathcal{G})$ *iff* $\models_{\mathsf{GK}^{\mathsf{F}}_\Box} \mathrm{I}(\mathcal{G})$.

Proof. If $\vdash_{\mathsf{HGK}_\Box} \mathrm{I}(\mathcal{G})$, then $\vdash_{\mathsf{GGK}_\Box} \mathcal{G}$, since (1) all the axioms of $\mathsf{HGK}_\Box$ are derivable in $\mathsf{GGK}_\Box$ and the rules are admissible, and (2) $\vdash_{\mathsf{GGK}_\Box} \mathcal{G}$ iff $\vdash_{\mathsf{GGK}_\Box}$ $I(\mathcal{G})$ (proved as for GG in [16]). Moreover, to show that $\vdash_{\mathsf{GGK}_\Box} \mathcal{G}$ implies $\models_{\mathsf{GK}_\Box}$ $\mathrm{I}(\mathcal{G})$, we need only show that the rules of $\mathsf{GGK}_\Box$ are sound with respect to the semantics, the new case of ($\Box$) following exactly as in the proof of Theorem 3. All other implications follow from Theorem 1 (also proved using the completeness of $\mathsf{SGK}_\Box$ as Corollary 1). □

We now show that cut-elimination holds for $\mathsf{GGK}_\Box$, i.e., that there is a constructive procedure for transforming a derivation of a hypersequent $\mathcal{G}$ in this calculus into a derivation of $\mathcal{G}$ with no applications of (CUT). We write $d \vdash_{\mathsf{S}} X$ to denote that d is a derivation of some structure X in a calculus S and $|d|$ for the height of the derivation considered as a tree. We also recall that the principal formula of an application of a rule is the distinguished formula in the conclusion and that the cut-formula of an application of (CUT) is the formula appearing in the premises but not the conclusion.

Theorem 7. *Cut-elimination holds for* $\mathsf{GGK}_\Box$.

Proof. Let $\mathsf{GGK}^\circ_\Box$ be $\mathsf{GGK}_\Box$ with (CUT) removed. Then to establish cut-elimination for $\mathsf{GGK}_\Box$ it is sufficient to give a constructive proof of the following:

Claim. If $d_1 \vdash_{\mathsf{GGK}^\circ_\Box} [\varGamma_i, [A]^{\lambda_i} \Rightarrow \varDelta_i]_{i=1}^{n}$ and $d_2 \vdash_{\mathsf{GGK}^\circ_\Box} \mathcal{H} \mid [\varPi_j \Rightarrow A]_{j=1}^{m}$,
then $\vdash_{\mathsf{GGK}^\circ_\Box} \mathcal{H} \mid [\varGamma_i, \varPi_j^{\lambda_i} \Rightarrow \varDelta_i]_{i=1\ldots n}^{j=1\ldots m}$.

We proceed by induction on the lexicographically ordered pair $\langle |A|, |d_1| + |d_2| \rangle$. If the last step in either derivation is an initial hypersequent, then the result follows almost immediately. Also, if the last step in either derivation is not ($\Box$) or does not have the cut-formula as the principal formula, then the result follows by applications of the induction hypothesis to the premises and applications of the same rule and structural rules. Suppose for example that one of the derivations ends with an application of (COM) (the other derivation may end with ($\Box$)):

$$\frac{\mathcal{G} \mid \Gamma_1', \Gamma_2', [A]^{\lambda_1'+\lambda_2'} \Rightarrow \Delta_1 \quad \mathcal{G} \mid \Gamma_1'', \Gamma_2'', [A]^{\lambda_1''+\lambda_2''} \Rightarrow \Delta_2}{\mathcal{G} \mid \Gamma_1', \Gamma_1'', [A]^{\lambda_1'+\lambda_1''} \Rightarrow \Delta_1 \mid \Gamma_2', \Gamma_2'', [A]^{\lambda_2'+\lambda_2''} \Rightarrow \Delta_2}$$

where $\mathcal{G} = [\Gamma_i, [A]^{\lambda_i} \Rightarrow \Delta_i]_{i=3}^{n}$. Then by the induction hypothesis twice:

$$\vdash_{\mathsf{GGK}^{\circ}_{\Box}} \mathcal{H}' \mid [\Gamma_1', \Gamma_2', \Pi_j^{\lambda_1'+\lambda_2'} \Rightarrow \Delta_1]_{j=1}^{m} \quad \vdash_{\mathsf{GGK}^{\circ}_{\Box}} \mathcal{H}' \mid [\Gamma_1'', \Gamma_2'', \Pi_j^{\lambda_1''+\lambda_2''} \Rightarrow \Delta_2]_{j=1}^{m}$$

where $\mathcal{H}' = \mathcal{H} \mid [\Gamma_i, \Pi_j^{\lambda_i} \Rightarrow \Delta_i]_{i=3\ldots n}^{j=1\ldots m}$. The required hypersequent:

$$\mathcal{H}' \mid [\Gamma_1', \Pi_j^{\lambda_1'+\lambda_1''}, \Gamma_1'' \Rightarrow \Delta_1]_{j=1}^{m} \mid [\Gamma_2', \Pi_j^{\lambda_2'+\lambda_2''}, \Gamma_2'' \Rightarrow \Delta_2]_{j=1}^{m}$$

is derivable by repeated applications of (COM), (EC), and (EW).

If a distinguished occurrence of A is the principal formula in both derivations and of the form $B \wedge C$, $B \vee C$, or $B \to C$, then we can first use the induction hypothesis applied to the premises in one derivation and the conclusion in the other, and then apply the induction hypothesis again with cut-formulas B and C of smaller complexity. The result follows using applications of (EC) and/or (EW) as required. Consider then the hardest case where both derivations d_1 and d_2 end as follows with an application of ($\Box$) and A is of the form $\Box B$:

$$\dfrac{\dfrac{\vdots}{\Gamma_1, [B]^{\lambda_1} \Rightarrow\mid \Gamma_2, [B]^{\lambda_2} \Rightarrow C}}{\Box\Gamma_1, [\Box B]^{\lambda_1} \Rightarrow\mid \Box\Gamma_2, [\Box B]^{\lambda_2} \Rightarrow \Box C}\,(\Box) \qquad \dfrac{\dfrac{\vdots}{\Sigma \Rightarrow\mid \Pi \Rightarrow B}}{\Box\Sigma \Rightarrow\mid \Box\Pi \Rightarrow \Box B}\,(\Box)$$

Then since $|B| < |\Box B|$, we can apply the induction hypothesis to the $\mathsf{GGK}^{\circ}_{\Box}$-derivable hypersequents $\Gamma_1, [B]^{\lambda_1} \Rightarrow\mid \Gamma_2, [B]^{\lambda_2} \Rightarrow C$ and $\Sigma \Rightarrow\mid \Pi \Rightarrow B$ to obtain a $\mathsf{GGK}^{\circ}_{\Box}$-derivation of $\Sigma \Rightarrow\mid \Gamma_1, \Pi^{\lambda_1} \Rightarrow\mid \Gamma_2, \Pi^{\lambda_2} \Rightarrow C$. Hence by an application of the derived rule $(\Box)^*$, we obtain a $\mathsf{GGK}^{\circ}_{\Box}$-derivation ending with $\Box\Sigma \Rightarrow\mid \Box\Gamma_1, \Box\Pi^{\lambda_1} \Rightarrow\mid \Box\Gamma_2, \Box\Pi^{\lambda_2} \Rightarrow \Box C$ as required. □

Concluding Remark. In this paper, we have introduced sequent of relations and hypersequent calculi and established PSPACE completeness for a basic Gödel modal logic. Our broader goal is to extend these results to classes of modal fuzzy logics. Certain Gödel modal logics with accessibility relations that are transitive, reflexive, symmetric, etc. (giving e.g. $\mathsf{K4}$ and $\mathsf{S4}$ versions of the logic) are axiomatized in [6] and in these cases, it is a straightforward task to develop corresponding hypersequent calculi extending $\mathsf{GGK}_{\Box}$ that admit cut-elimination. However, a general treatment incorporating more complicated modal conditions will require more expressive formalisms such as display calculi or labelled sequents/tableaux as is already the case for classical modal logics. Another task, important in connection with fuzzy description logics, is to develop calculi to deal with logics equipped also with the modality $\Diamond$. Although similar methods to those developed for the single modality should work, the logics based on standard and fuzzy Kripke frames diverge and require different rules.

References

1. Avron, A.: A constructive analysis of RM. Journal of Symbolic Logic 52(4), 939–951 (1987)
2. Avron, A.: Hypersequents, logical consequence and intermediate logics for concurrency. Annals of Mathematics and Artificial Intelligence 4(3–4), 225–248 (1991)
3. Baaz, M., Ciabattoni, A., Fermüller, C.G.: Hypersequent calculi for Gödel logics: a survey. Journal of Logic and Computation 13, 1–27 (2003)
4. Baaz, M., Fermüller, C.G.: Analytic calculi for projective logics. In: Murray, N.V. (ed.) TABLEAUX 1999. LNCS (LNAI), vol. 1617, pp. 36–50. Springer, Heidelberg (1999)
5. Bou, F., Esteva, F., Godo, L., Rodríguez, R.: On the minimum many-valued logic over a finite residuated lattice (manuscript)
6. Caicedo, X., Rodríguez, R.: A Gödel modal logic (manuscript)
7. Chagrov, A., Zakharyaschev, M.: Modal Logic. Oxford University Press, Oxford (1996)
8. Ciabattoni, A., Metcalfe, G., Montagna, F.: Adding modalities to MTL and its extensions. In: Proceedings of the 26th Linz Symposium (to appear)
9. Dummett, M.: A propositional calculus with denumerable matrix. Journal of Symbolic Logic 24, 97–106 (1959)
10. Dyckhoff, R.: A deterministic terminating sequent calculus for Gödel-Dummett logic. Logic Journal of the IGPL 7(3), 319–326 (1999)
11. Fitting, M.C.: Many-valued modal logics. Fundamenta Informaticae 15(3-4), 235–254 (1991)
12. Fitting, M.C.: Many-valued modal logics II. Fundamenta Informaticae 17, 55–73 (1992)
13. Gödel, K.: Zum intuitionisticschen Aussagenkalkül. Anzeiger Akademie der Wissenschaften Wien, mathematisch-naturwiss. Klasse 32, 65–66 (1932)
14. Hájek, P.: Metamathematics of Fuzzy Logic. Kluwer, Dordrecht (1998)
15. Hájek, P.: Making fuzzy description logic more general. Fuzzy Sets and Systems 154(1), 1–15 (2005)
16. Metcalfe, G., Olivetti, N., Gabbay, D.: Proof Theory for Fuzzy Logics. Applied Logic, vol. 36. Springer, Heidelberg (2009)
17. Priest, G.: Many-valued modal logics: a simple approach. Review of Symbolic Logic 1, 190–203 (2008)
18. Sonobe, O.: A Gentzen-type formulation of some intermediate propositional logics. Journal of Tsuda College 7, 7–14 (1975)
19. Straccia, U.: Reasoning within fuzzy description logics. Journal of Artificial Intelligence Research 14, 137–166 (2001)
20. Wolter, F.: Superintuitionistic companions of classical modal logics. Studia Logica 58(2), 229–259 (1997)
21. Zhang, Z., Sui, Y., Cao, C., Wu, G.: A formal fuzzy reasoning system and reasoning mechanism based on propositional modal logic. Theoretical Computer Science 368(1-2), 149–160 (2006)

Generic Modal Cut Elimination Applied to Conditional Logics

Dirk Pattinson[1,⋆] and Lutz Schröder[2,⋆⋆]

[1] Department of Computing, Imperial College London
[2] DFKI Bremen and Department of Computer Science, Universität Bremen

Abstract. We develop a general criterion for cut elimination in sequent calculi for propositional modal logics, which rests on absorption of cut, contraction, weakening and inversion by the purely modal part of the rule system. Our criterion applies also to a wide variety of logics outside the realm of normal modal logic. We give extensive example instantiations of our framework to various conditional logics. For these, we obtain fully internalised calculi which are substantially simpler than those known in the literature, along with leaner proofs of cut elimination and complexity. In one case, conditional logic with modus ponens and conditional excluded middle, cut elimination and complexity are explicitly stated as open in the literature.

1 Introduction

Cut elimination, originally invented by Gentzen [5], is one of the core concepts of proof theory and plays a major role in particular for algorithmic aspects of logic, including the subformula property, the complexity of automated reasoning and, via interpolation, modularity issues. The large number of logical calculi that are currently in use, in particular in various areas of computer science, motivates efforts to define families of sequent calculi that cover a variety of logics and admit uniform proofs of cut elimination, enabled by suitable sufficient conditions. Here, we present such a method for modal sequent calculi that applies to possibly non-normal normal modal logics, which appear, e.g. in concurrency and knowledge representation. We use a separation of the modal calculi into a fixed underlying propositional part and a modal part; the core of our criterion is absorption of cut by the modal rules. This concept generalises the notion of resolution closed rule set [9,12], dropping the assumption that the logic at hand is rank-1, i.e. axiomatised by formulas in which the nesting depth of modal operators is uniformly equal to 1 (such as K).

Our method is reasonably simple and intuitive, and nevertheless applies to a wide range of modal logics. While we use normal modal logics such as K and T as running examples to illustrate our concepts at the time of introduction, our

⋆ Partially supported by EPSRC grant EP/F031173/1.

⋆⋆ Work performed as part of the DFG project *Generic Algorithms and Complexity Bounds in Coalgebraic Modal Logic* (SCHR 1118/5-1).

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 280–294, 2009.

main applications are conditional logics, which have a binary modal operator read as a non-monotonic implication (unlike default logics, conditional logics allow nested non-monotonic implications). In particular, we prove cut-elimination (hence, since the generic systems under consideration are analytic, the subformula property) for the conditional logics CK, CKMP, CKCEM, and CKMPCEM using our generic procedure. An easy analysis of proof search in the arising cut-free calculi moreover establishes that the satisfiability problem of each of these logics is in *PSPACE*. This is a tight bound for CK and CKMP, whereas the provability problem in extensions of CKCEM can be solved in *coNP*, as we show by a slightly adapted algorithmic treatment of our calculus using a dynamic programming approach in the spirit of [13]. We point out that while (different) cut-free labelled sequent calculi for CK, CKMP, CKCEM, and some further conditional logics, as well as the ensuing upper complexity bounds, have previously been presented by Olivetti et al., the corresponding issues for CKMPCEM have explicitly been left as open problems [8].

Related work. A set of sufficient conditions for a sequent calculus to admit cut elimination and a subsequent analysis of the complexity *of cut elimination* (not proof search) is presented in [10]. The range of application of this method is very wide and encompasses, e.g. first-order logic, the modal logic $S4$, linear logic, and intuitionistic propositional logic. This generality is reflected in the fact that the method as a whole is substantially more involved than ours. A simpler method for a different and comparatively restrictive class of calculi, so-called canonical calculi, is considered in [1]; this method does not apply to typical modal systems, as it considers only so-called *canonical* rules, i.e., left and right introduction rules for connectives which permit adding a common context simultaneously in the premise and the conclusion. (In fact, it might be regarded as the essence of modal logic that its rules fail to be canonical, e.g. the necessitation rule $A/\Box A$ does not generalise to $\Gamma, A/\Gamma, \Box A$ for a sequent Γ.) Moreover, the format of the rules in *op.cit.* does not allow for the introduction of more than one occurrence of a logical connective, which is necessary even for the most basic modal logics. The same applies to [4]. In [3], logical rules are treated on an individual basis, which precludes the treatment of cuts between two rule conclusions. Overall, our notion of absorption is substantially more general when compared to similar notions in the papers discussed above, which stipulate that cuts between left and right rules for the same connective are absorbed by structural rules. In our own earlier work [9], we have considered a special case of the method presented here in the restricted context of *rank-1* logics; in particular, these results did not cover logics such as $K4$, CKMP, or CKMPCEM.

2 Preliminaries and Notation

A *modal similarity type* (or modal signature) is a set Λ of modal operators with associated arities that we keep fixed throughout the paper. Given a set V of propositional variables, the set $\mathcal{F}(\Lambda)$ of Λ-formulas is given by the grammar

$$\mathcal{F}(\Lambda) \ni A, B ::= \bot \mid p \mid \neg A \mid A \wedge B \mid \heartsuit(A_1, \dots, A_n)$$

where $p \in V$ and $\heartsuit \in \Lambda$ is n-ary. We use standard abbreviations of the other propositional connectives $\top$, $\vee$ and $\rightarrow$. A *Λ-sequent* is a finite multiset of Λ-formulas, and the set of Λ-sequents is denoted by $\mathsf{S}(\Lambda)$. We write the multiset union of Γ and Δ as Γ, Δ and identify a formula $A \in \mathcal{F}(\Lambda)$ with the singleton sequent containing only A. If $S \subseteq \mathcal{F}(\Lambda)$ is a set of formulas, then an *S-substitution* is a mapping $\sigma : V \rightarrow S$. We denote the result of uniformly substituting $\sigma(p)$ for p in a formula A by $A\sigma$. This extends pointwise to Λ-sequents so that $\Gamma\sigma = A_1\sigma, \ldots, A_n\sigma$ if $\Gamma = A_1, \ldots, A_n$. If $S \subseteq \mathcal{F}(\Lambda)$ is a set of Λ-formulas and $A \in \mathcal{F}(\Lambda)$, we say that A is a *propositional consequence* of S if there exist $A_1, \ldots, A_n \in S$ such that $A_1 \wedge \cdots \wedge A_n \rightarrow A$ is a substitution instance of a propositional tautology. We write $S \vdash_{\mathsf{PL}} A$ if A is a propositional consequence of S and $A \vdash_{\mathsf{PL}} B$ for $\{A\} \vdash_{\mathsf{PL}} B$ for the case of single formulas.

3 Modal Deduction Systems

To facilitate the task of comparing the notion of provability in both Hilbert and Gentzen type proof systems, we introduce the following notion of a proof rule that can be used, without any modifications, in both systems.

Definition 1. *A Λ-*rule *is of the form $\frac{\Gamma_1,\ldots,\Gamma_n}{\Gamma_0}$ where $n \geq 0$ and $\Gamma_0, \ldots, \Gamma_n$ are Λ-sequents. The sequents $\Gamma_1, \ldots, \Gamma_n$ are the* premises *of the rule and Γ_0 its* conclusion. *A rule $\frac{}{\Gamma_0}$ without premises is called a Λ-*axiom, *which we denote by just its conclusion, Γ_0. A* rule set *is just a set of Λ-rules, and we say that a rule set* R *is* substitution closed, *if $\Gamma_1\sigma \ldots \Gamma_n\sigma/\Gamma_0\sigma \in \mathsf{R}$ whenever $\Gamma_1 \ldots \Gamma_n/\Gamma_0 \in \mathsf{R}$ and $\sigma : V \rightarrow \mathcal{F}(\Lambda)$ is a substitution.*

In view of the sequent calculi that we introduce later, we read sequents disjunctively. Consequently, a rule $\Gamma_1, \ldots, \Gamma_n/\Gamma_0$ can be used to prove the disjunction Γ_0, provided that $\bigvee \Gamma_i$ is provable, for all $1 \leq i \leq n$. We emphasise that a rule is an expression of the object language, i.e. it does not contain meta-linguistic variables. As such, it represents a specific deduction step rather than a family of possible deductions, which helps to economise on syntactic categories. In our examples, concrete rule sets are presented as instances of rule schemas.

Example 2. For the modal logics K, $K4$ and T, we fix the modal signature $\Lambda = \{\Box\}$ consisting of a single modal operator $\Box$ with arity one. The language of conditional logic is given by the similarity type $\Lambda = \{\Rightarrow\}$ where the conditional arrow $\Rightarrow$ has arity 2. We use infix notation and write $A \Rightarrow B$ instead of $\Rightarrow (A, B)$ for $A, B \in \mathcal{F}(\Lambda)$. Deduction over modal and conditional logics are governed by the following rule sets:

1. The rule set K associated to the modal logic K consists of all instances of the necessitation rule (N) and the distribution axiom (D) below.

$$(\mathsf{N})\frac{A}{\Box A} \qquad (\mathsf{D})\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B) \qquad (\mathsf{4})\Box\Box A \rightarrow \Box A \qquad (\mathsf{R})\Box A \rightarrow A$$

The rule sets for T and $K4$ arise by extending this set with the reflexivity axiom (R) and the $(\mathsf{4})$-axiom, respectively. We reserve the name (T) for the reflexivity rule in a cut-free system.

2. Conditional logic, e.g. the system CK of [2] is axiomatised by the rule set that consists of all instances of (RCEA) on the left, and (RCK) on the right below:

$$\frac{A \leftrightarrow A'}{(A \Rightarrow B) \leftrightarrow (A' \Rightarrow B)} \qquad \frac{B_1 \land \dots \land B_n \to B}{(A \Rightarrow B_1) \land \dots \land (A \Rightarrow B_n) \to (A \Rightarrow B)}$$

As additional axioms, we consider

$$(\mathsf{ID}) A \Rightarrow A \qquad (\mathsf{MP})(A \Rightarrow B) \to (A \to B) \qquad (\mathsf{CEM})(A \Rightarrow B) \lor (A \Rightarrow \neg B)$$

that induce extensions of CK that we denote by juxtaposition of the respective axioms, e.g. $\mathsf{CKMPCEM}$ contains the rules for CK and the axioms (MP) and (CEM).

Rules with more than one premise arise through saturation of a given rule set under cut that, e.g. leads to the rules (CK_g) and (MP_g) presented in Section 6.

The notion of deduction in modal Hilbert systems then takes the following form.

Definition 3. *Suppose* R *is a set of rules. The set of* R*-derivable formulas in the Hilbert-system given by* R *is the least set of formulas that*

- *contains* $A\sigma$ *whenever* A *is a propositional tautology and* σ *is a substitution*
- *contains* B *whenever it contains* A *and* $A \to B$
- *contains* $\bigvee \Gamma_0$ *whenever it contains* $\bigvee \Gamma_1, \dots, \bigvee \Gamma_n$ *and* $\frac{\Gamma_1 \dots \Gamma_n}{\Gamma_0} \in \mathsf{R}$.

We write $\mathsf{HR} \vdash A$ *if* A *is* R*-derivable.*

In other words, the set of derivable formulas is the least set that contains propositional tautologies, is closed under uniform substitution, modus ponens and application of rules. We will later consider Hilbert systems that induce the same provability predicate based on the following notion of admissibility.

Definition 4. *A rule set* R' *is* admissible *in* HR *if* $\mathsf{HR} \vdash A \iff \mathsf{H}(\mathsf{R} \cup \mathsf{R}') \vdash A$ *for all formulas* $A \in \mathcal{F}(\Lambda)$*. Two rule sets* R, R' *are* equivalent *if* R *is admissible in* HR' *and* R' *is admissible in* HR.

In words, R' is admissible in HR if adding the rules R' to those of R leaves the set of provable formulas unchanged. We note the following trivial, but useful consequence of admissibility.

Lemma 5. $\mathsf{HR} \vdash A$ *iff* $\mathsf{HR}' \vdash A$ *if* R *and* R' *are equivalent and* $A \in \mathcal{F}(\Lambda)$.

The next proposition is concerned with the structure of proofs in Hilbert systems and is the key for proving equivalence of Hilbert and Gentzen-type systems.

Proposition 6. *The set* $\mathsf{HT}(\mathsf{R}) = \{A \in \mathcal{F}(\Lambda) \mid \mathsf{HR} \vdash A\}$ *is the smallest set* S *of formulas that contains a formula* $A \in \mathcal{F}(\Lambda)$ *whenever there are rules* $\Theta_1/\Gamma_1, \dots, \Theta_n/\Gamma_n \in \mathsf{R}$ *and substitutions* $\sigma_1, \dots, \sigma_n : V \to \mathcal{F}(\Lambda)$ *such that* $\bigvee \Delta\sigma_i \in S$ *for all* $\Delta \in \Theta_i$ $(i = 1, \dots, n)$ *and* $\{\bigvee \Gamma_1\sigma, \dots, \bigvee \Gamma_n\sigma\} \vdash_{\mathsf{PL}} A$.

In other words, in a modal Hilbert system, each provable formula is a propositional consequence of rule conclusions with provable premises. This result forms the basis of our comparison of Hilbert and Gentzen systems, and we show that cut elimination essentially amounts to the fact that – in the corresponding Hilbert system – each valid formula is a consequence of a *single* rule conclusion with provable premise.

We now set the stage for sequent systems that we are going to address in the remainder of the paper. The notion of derivability in the sequent calculus associated with a rule set R is formulated parametric in terms of a set X of additional rules that will later be instantiated with relativised versions of cut, weakening, contraction and inversion.

Definition 7. *Suppose* R *and* X *are sets of Λ-rules. The set of* $\mathsf{RC}+\mathsf{X}$*-derivable sequents in the Gentzen-system given by* R *is the least set of sequents that*

- *contains $A, \neg A, \Gamma$ for all sequents $\Gamma \in \mathsf{S}(\Lambda)$ and formulas $A \in \mathcal{F}(\Lambda)$*
- *contains $\neg\bot, \Gamma$ for all $\Gamma \in \mathsf{S}(\Lambda)$*
- *is closed under instances of the rule schemas*

$$\frac{\Gamma, \neg A, \neg B}{\Gamma, \neg(A \wedge B)} \qquad \frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \wedge B} \qquad \frac{\Gamma, A}{\Gamma, \neg\neg A}$$

where $A \in \mathcal{F}(\Lambda)$ ranges over formulas and $\Gamma \subseteq \mathcal{F}(\Lambda)$ over multisets of formulas. We call the above rules the propositional rules *and the formula occurring in the conclusion but not in Γ* principal *in the respective rule.*

- *is closed under the rules in $\mathsf{R} \cup \mathsf{X}$, i.e. it contains Γ_0 whenever it contains $\Gamma_1, \ldots, \Gamma_n$ for $\frac{\Gamma_1 \ldots \Gamma_n}{\Gamma_0} \in \mathsf{R} \cup \mathsf{X}$.*

We write $\mathsf{GR} + \mathsf{X} \vdash \Gamma$ if Γ can be derived in this way and $\mathsf{GR} \vdash \Gamma$ if $\mathsf{X} = \emptyset$.

The set X of extra rules will later be instantiated with a relativised version of the cut rule and additional axioms that locally capture the effect of weakening, contraction and inversion, applied to rule premises. This allows to formulate *local* conditions for the admissibility of cut that can be checked on a per-rule basis.

Many other formulations of sequent systems only permit axioms of the form $\Gamma, p, \neg p$ where $p \in V$ is a propositional atom. The reason for being more liberal here is that this makes it easier to prove admissibility of uniform substitution, at the expense of loosing depth-preserving admissibility of structural rules. We come back to this matter in Remark 12.

The following proposition is readily established by an induction on the provability predicate $\mathsf{RH} \vdash$.

Proposition 8. *Suppose $\Gamma \in \mathsf{S}(\Lambda)$ is a sequent. Then $\mathsf{RH} \vdash \bigvee \Gamma$ if $\mathsf{RG} \vdash \Gamma$.*

The remainder of the paper is concerned with the converse of the above proposition, which relies on specific properties of the rule set R.

4 Generic Modal Cut Elimination

In order to establish the converse of Proposition 8 we need to establish that the cut rule is admissible in the Gentzen system GR defined by the ruleset R. Clearly, we cannot expect that cut elimination holds in general: it is well known (and easy to check) that the sequent system arising from the rule set consisting of all instances of (N) and (D), presented in Example 2 does *not* enjoy cut elimination. In other words, we have to look for constructions that allow us to transform a given rule set into one for which cut elimination holds. The main result of our analysis is that cut elimination holds if the rule set under consideration satisfies four crucial requirements that are local in the sense that they can be checked on a per-rule basis without the need of carrying out a fully-fledged cut-elimination proof: absorption of weakening, contraction, inversion and cut.

The first three properties can be checked for each rule individually and amount to the admissibility of the respective principle, and the last requirement amounts to the possibility of eliminating cut between a pair of rule conclusions. We emphasise that these properties can be checked locally for the modal rules, and cut elimination will follow automatically. It is not particularly surprising that cut elimination holds under these assumptions. However, isolating the four conditions above provides us with means to convert a modal Hilbert system into an equivalent cut-free sequent calculus. We now introduce relativised versions of the structural rules that will be the main tool in the proof of cut elimination.

Definition 9. *Suppose Γ is a Λ-sequent and let $\mathsf{A}(\Gamma)$ consist of the axioms*

- Γ, A *for all* $A \in \mathcal{F}(\Lambda)$
- Δ, A *if* $\Gamma = \Delta, A, A$ *for some* $\Delta \in \mathsf{S}(\Lambda), A \in \mathcal{F}(\Lambda)$
- Δ, A *if* $\Gamma = \Delta, \neg\neg A$ *for some* $\Delta \in \mathsf{S}(\Lambda), A \in \mathcal{F}(\Lambda)$
- $\Delta, \neg A_1, \neg A_2$ *if* $\Gamma = \Delta, \neg(A_1 \wedge A_2)$ *for some* $\Delta \in \mathsf{S}(\Lambda)$, $A_1, A_2 \in \mathcal{F}(\Lambda)$
- Δ, A_i *for* $i = 1, 2$ *if* $\Gamma = \Delta, (A_1 \wedge A_2)$ *for some* $\Delta \in \mathsf{S}(\Lambda)$, $A_1, A_2 \in \mathcal{F}(\Lambda)$

We say that a rule set R absorbs the structural rules *if*

$$\mathsf{GR} + \mathsf{A}(\Gamma_1) \cup \cdots \cup \mathsf{A}(\Gamma_n) \vdash \Gamma$$

for all $\frac{\Gamma_1 \ldots \Gamma_n}{\Gamma_0} \in \mathsf{R}$ *and all* $\Gamma \in \mathsf{A}(\Gamma_0)$.

In other words, a deduction step that applies weakening, contraction or inversion to a rule conclusion can be replaced by a (possibly different) rule where the corresponding structural rules are applied to the premises. We discuss a number of standard examples before stating that absorption of the structural rules implies their admissibility.

Example 10. The rule sets containing all instances of either of the following rule schemas (K), (T) and (K4)

$$\frac{\neg A_1, \ldots, \neg A_n, A_0}{\neg\Box A_1, \ldots, \neg\Box A_n, \Box A_0, \Gamma} \qquad \frac{\neg A, \neg\Box A, \Gamma}{\neg\Box A, \Gamma} \qquad \frac{\neg A_1, \neg\Box A_1, \ldots, \neg A_n, \neg\Box A_n, B}{\neg\Box A_1, \ldots, \neg\Box A_n, \Box B, \Gamma}$$

absorbs the structural rules. We note that (K) absorbs weakening due to the presence of Γ in the conclusion, and the absorption of contraction in (T) and (K4) is a consequence of the presence of the negated $\Box$-formulas in the premise. The absorption of inversion in a consequence of the weakening context Γ in (K) and (K4) and implied by duplicating the context Γ in (T). On the other hand, the rule sets defined by

$$\frac{\neg A_1, \dots, \neg A_n, A_0}{\neg\Box A_1, \dots, \neg\Box A_n, \Box A_0} \qquad\qquad \frac{\neg A, \Gamma}{\neg\Box A, \Gamma}$$

fail to absorb the structural rules: the rule on the left fails to absorb weakening, whereas the right-hand rule does not absorb contraction.

It should be intuitively clear that absorption of structural rules implies their admissibility, which we establish next.

Proposition 11. *Suppose* R *absorbs the structural rules. Then all instances of the rule schemas of weakening, contraction and inversion*

$$\frac{\Gamma}{\Gamma, A} \qquad \frac{\Gamma, A, A}{\Gamma, A} \qquad \frac{\Gamma, \neg\neg A}{\Gamma, A} \qquad \frac{\Gamma, \neg(A_1 \wedge A_2)}{\Gamma, \neg A_1, \neg A_2} \qquad \frac{\Gamma, A_1 \wedge A_2}{\Gamma, A_i}(i = 1, 2)$$

where $\Gamma \in \mathsf{S}(\Lambda)$ *and* $A, A_1, A_2 \in \mathcal{F}(\Lambda)$ *are admissible in* GR.

Remark 12

1. The main purpose for introducing the notion of absorption of structural rules (Definition 9) is to have a handy criterion that guarantees admissibility of the structural rules (Proposition 11). Our definition offers a compromise between generality and simplicity. In essence, a rule set absorbs structural rules, if an application of weakening, contraction or inversion can be pushed up one level of the proof tree. A weaker version of Definition 9 would require that an application of weakening, contraction or inversion to a rule conclusion can be replaced by a sequence of deduction steps where the structural rule in question can not only be applied to the premises of the rule, but also freely anywhere else, provided that these additional applications are smaller in a well-founded ordering. However, we are presently not aware of any examples where this extra generality would be necessary.

2. In many Sequent systems, the statement of Proposition 11 can be strengthened to say that weakening, contraction and inversion are depth-preserving admissible, i.e. does not increase the height of the proof tree. This is in general false for the systems considered here as axioms are of the form $A, \neg A, \Gamma$ for $A \in \mathcal{F}(\Lambda)$ and, for instance, $(A \wedge B), \neg(A \wedge B)$ is derivable with a proof of height one (being an axiom), but, e.g. $A \wedge B, \neg A, \neg B$ cannot be established by a proof of depth one (not being an axiom). It is easy to see that weakening, inversion and contraction are in fact depth-preserving admissible if only atomic axioms of the form $p, \neg p, \Gamma$ are allowed, for $p \in V$ a propositional variable. The more general form of axioms adopted in this paper allows us to simplify many constructions as we do not have to consider a congruence rule explicitly which would allow us to prove (rather than to assume as axioms) sequents of the form $\Box A, \neg\Box A, \Gamma$.

Having dealt with the structural rules, we now address our main concern: the admissibility of the cut rule. In contrast to the absorption of structural rules, we need one additional degree of freedom in that we need to allow ourselves to apply cut to a structurally smaller formula.

Definition 13. *The* size *of a formula $A \in \mathcal{F}(\Lambda)$ is given inductively by* $\mathsf{size}(p) = \mathsf{size}(\bot) = 1$, $\mathsf{size}(A \wedge B) = \mathsf{size}(A \vee B) = 1 + \mathsf{size}(A) + \mathsf{size}(B)$ *and, for the modal case,* $\mathsf{size}(\heartsuit(A_1, \dots, A_n)) = 1 + \mathsf{size}(A_1) + \dots + \mathsf{size}(A_n)$.

A ruleset R absorbs cut, *if for all rules* $(r_1)\frac{\Gamma_1, \dots, \Gamma_n}{A, \Gamma_0}$, $(r_2)\frac{\Delta_1, \dots, \Delta_k}{\neg A, \Delta_0} \in \mathsf{R}$

$$\mathsf{GR} + \mathsf{Cut}(A, r_1, r_2) \vdash \Gamma_0, \Delta_0$$

where $\mathsf{Cut}(A, r_1, r_2)$ *consists of all instances of the rule schemas*

$$\frac{\Gamma, C \quad \Delta, \neg C}{\Gamma, \Delta} \quad \frac{\Gamma}{\Gamma, A} \qquad \frac{\Gamma, A, A}{\Gamma, A} \quad \frac{\Gamma, \neg\neg A}{\Gamma, A} \quad \frac{\Gamma, \neg(A_1 \wedge A_2)}{\Gamma, \neg A_1, \neg A_2} \quad \frac{\Gamma, A_1 \wedge A_2}{\Gamma, A_i}$$

where $\mathsf{size}(C) < \mathsf{size}(A)$ *in the leftmost rule and $i = 1, 2$ in the rightmost schema, together with the axioms $\Gamma_1, \dots, \Gamma_n, \Delta_1, \dots, \Delta_k$ and all sequents of the form Γ, Δ where $\Gamma, \Delta \in \mathsf{S}(\Lambda)$ and, for some $B \in \mathcal{F}(\Lambda)$,*

- Γ, B *and* $\Delta, \neg B \in \{\Gamma_1, \dots, \Gamma_n, \Delta_1, \dots, \Delta_k\}$, *or*
- $\Gamma, B = \Gamma_0, A$ *and* $\Delta, \neg B \in \{\Delta_1, \dots, \Delta_k\}$, *or*
- $\Gamma, B = \Delta_0, \neg A$ *and* $\Delta, \neg B \in \{\Gamma_1, \dots, \Gamma_n\}$.

A rule set that absorbs structural rules and the cut rule is called absorbing*.*

The intuition behind the above definition is similar to that of absorption of structural rules, but we have two additional degrees of freedom: we can not only apply the cut rule to rule premises, but we can moreover freely use both cut on structurally smaller formulas and the structural rules. This allows us to use the standard argument, a double induction on the structure of the cut formula and the size (or height) of the proof tree, to establish cut elimination. This is carried out in the proof of the next theorem.

Theorem 14. *Suppose* R *is absorbing. Then the cut rule*

$$\frac{\Gamma, A \qquad \Delta, \neg A}{\Gamma, \Delta}$$

is admissible in GR*.*

The proof proceeds by a double induction on the size of the cut formula and the size of the proof tree, and analyse all possible ways in which the cut rule can be applied. The case of cuts arising between conclusions of modal rules follows from absorption of cut. Cuts between conclusions of a modal and a propositional rule can be eliminated by using the absorption of structural rules.

We illustrate the preceding theorem by using it to derive the well-known fact that cut-elimination holds for the modal logics K, K4 and T and use it to derive cut-elimination for various conditional logics in Section 6.

Example 15. The rule sets K, K4 and T are absorbing. We have already seen that they absorb weakening, contraction and inversion in Example 10 so everything that remains to be seen is that they also absorb cut. For (K), we need to apply cut to a formula of smaller size. For the two instances

$$(r_1)\frac{\neg A_1, \dots, \neg A_n, A_0}{\neg\Box A_1, \dots, \neg\Box A_n, \Box A_0, \Gamma} \qquad (r_2)\frac{\neg B_1, \dots, \neg B_k, B_0}{\neg\Box B_1, \dots, \neg\Box B_k, \Box B_0, \Delta}$$

we need to consider, up to symmetry, the cases $A_i = B_0$, $\Box A_i \in \Delta$ and $\neg\Box A_0 \in \Delta$, for $i = 1, \ldots, n$. Here, we only treat the first case for $i = 1$ where we have to show that $\neg\Box A_2, \ldots, \neg\Box A_n, \Box A_0, \neg\Box B_1, \ldots, \neg\Box B_k, \Gamma, \Delta$ is derivable from $\mathsf{GR} + \mathsf{Cut}(\Box A_1, r_1, r_2)$, which follows as the latter system allows us to apply cut on $A_1 = B_0$. The case $\Box A_i \in \Delta$ and $\neg\Box A_0 \in \Delta$ are straight forward.

The argument to show that $(\mathsf{K4})$ is absorbing is similar, and uses an additional (admissible) instance of cut on a formula of smaller size and contraction. For (T) we only consider instances of cut between two conclusions of

$$(r_1)\frac{\neg A, \neg\Box A, \Gamma}{\neg\Box A, \Gamma} \qquad (r_2)\frac{\neg B, \neg\Box B, \Delta}{\neg\Box B, \Delta}$$

of the T-rule. We only demonstrate the case $\Box A \in \Delta$. In this case, $\Delta = \Delta', \Box A$ and we have to show that $\neg\Box B, \Gamma, \Delta'$ can be derived in $\mathsf{Cut}(\Box A, r_1, r_2)$. The latter system allows us to cut $\neg\Box A$ between the conclusion of (T) on the left and the premise of the right hand rule, i.e., we have that $\mathsf{Cut}(\Box A, r_1, r_2) \vdash \neg B, \neg\Box B, \Gamma, \Delta')$ and an application of (T) now gives derivability of $\neg\Box B, \Gamma, \Delta'$.

5 Equivalence of Hilbert and Gentzen Systems

We now investigate the relationship between provability in a Hilbert-system and provability in the associated Gentzen system. We note the following standard lemmas that we will use later on.

Lemma 16. *Suppose $A \in \mathcal{F}(\Lambda)$ is a propositional tautology. Then $\mathsf{GR} \vdash A$. If moreover R is closed under substitution, then $\mathsf{GR} \vdash \Gamma\sigma$ whenever $\mathsf{GR} \vdash \Gamma$ for all $\Gamma \in \mathsf{S}(\Lambda)$.*

Remark 17. Being able to prove the previous lemma is the main reason for formulating axioms as $A, \neg A, \Gamma$ where $A \in \mathcal{F}(\Lambda)$ rather than $p, \neg p, \Gamma$. Both formulations are equivalent if the modal congruence rule

$$\frac{A_1 \leftrightarrow A_1' \qquad \ldots \qquad A_n \leftrightarrow A_n'}{\heartsuit(A_1, \ldots, A_n) \to \heartsuit(A_1', \ldots, A_n')}$$

is admissible. However, Lemma 16 can be proved without the assumption that congruence is admissible using axioms of the form $A, \neg A, \Gamma$.

Theorem 18. *Suppose R is absorbing and substitution closed. Then $\mathsf{GR} \vdash \Gamma \iff \mathsf{HR} \vdash \bigvee \Gamma$ for all $\Gamma \in \mathsf{S}(\Lambda)$.*

Proof (Sketch). We only need to show the direction from right to left. Inductively assume that $\mathsf{HR} \vdash \bigvee \Gamma$ for $\Gamma \in \mathsf{S}(\Lambda)$. By Proposition 8 we have that there are rules Θ_i/Γ_i and substitutions σ_i, $i = 1, \ldots, n$ such that

- $\mathsf{HR} \vdash \Delta\sigma_i$ whenever $\Delta \in \Theta_i$ $(i = 1, \ldots, n)$
- $\{\bigvee \Gamma_1\sigma_1, \ldots, \bigvee \Gamma_n\sigma_n\} \vdash_{\mathsf{PL}} \bigvee \Gamma$.

By induction hypothesis, $\mathsf{GR} \vdash \Delta\sigma_i$ for all $i = 1, \ldots, n$ and $\Delta \in \Theta_i$. By Lemma 16 we have

$$\mathsf{GR} \vdash \bigvee \Gamma_1\sigma_1 \wedge \cdots \wedge \bigvee \Gamma_n\sigma_n \to \bigvee \Gamma.$$

The claim follows by applying cut, contraction and inversion.

The construction of an absorbing rule set from a given set of axioms and rules essentially boils down to adding the missing instances of cut, weakening, contraction and inversion to a given rule set. The soundness of this process is witnessed by the following two simple lemmas, which we use in this section to derive an absorbing rule set for K and to establish cut-elimination for a large range of conditional logics in the next section.

Lemma 19. *Suppose* $\Gamma_1, \ldots, \Gamma_n/\neg A, \Gamma_0$ *and* $\Delta_1, \ldots, \Delta_k/A, \Delta_0 \in \mathsf{R}$. *Then the rule* $\Gamma_1, \ldots, \Gamma_n, \Delta_1, \ldots, \Delta_k/\Gamma_0, \Delta_0$ *is admissible in* HR.

The same applies to instances of the structural rules of weakening, contraction and inversion. As we are extending the rule set while leaving the provability predicate in the Hilbert calculus unchanged, the following formulation is handy for our purposes – in particular it implies the fact that we can freely use structural rules both in the premise and conclusion.

Lemma 20. *Suppose that* $\Gamma_1, \ldots, \Gamma_n/\Gamma_0 \in \mathsf{R}$. *If* $\Delta_0, \ldots, \Delta_1 \in \mathsf{S}(\Lambda)$ *and both*

$$\{\bigvee \Delta_1, \ldots, \bigvee \Delta_k\} \vdash_{\mathsf{PL}} \bigvee \Gamma_i (1 \leq i \leq n) \textit{ and } \bigvee \Gamma_0 \vdash_{\mathsf{PL}} \bigvee \Delta_0$$

then the rule $\Gamma_1, \ldots, \Gamma_k/\Gamma_0$ *is admissible in* HR.

This gives us a recipe for constructing rule sets that absorb contraction and cut: simply add more rules according to the lemmas above. This will not change the notion of provability in the Hilbert system, but when this process terminates, the ensuing rule set will be absorbing and gives rise to a cut free sequent calculus.

Example 21 (Modal Logic K). In a Hilbert-style calculus, the axiomatisation of K is usually described in terms of the distribution axiom (which we view as a rule with empty premise) and the necessitation rule:

$$(\mathsf{D}) \quad \Box(A \to B) \to \Box A \to \Box B \qquad (\mathsf{N})\frac{A}{\Box A}$$

We first apply Lemma 19 to break the propositional connectives in the distribution axiom. We have that the axiom $\neg\Box(A \to B), \neg\Box A, \Box B$ is admissible by Lemma 20, and applying Lemma 19 to this axiom and the instance $A \to B/\Box(A \to B)$ of the necessitation rule gives admissibility of the all instances of

$$\frac{\neg A, B}{\neg\Box A, \Box B}$$

with the help of (admissible) propositional reasoning in the premise. The same procedure, applied to the instances

$$\frac{\neg A, B \to C}{\neg\Box A, \Box(B \to C)} \qquad \neg\Box(B \to C), \neg\Box B, \Box C$$

gives admissibility of the left hand rule below,

$$\frac{\neg A, \neg B, C}{\neg\Box A, \neg\Box B, \Box C} \qquad \frac{\neg A_1, \dots, \neg A_n, A_0}{\neg\Box A_1, \dots, \neg\Box A_n, \Box A_0, \Gamma}$$

and continuing in this way and absorbing weakening, we obtain admissibility of the rule on the right, where $\Gamma \in \mathsf{S}(\Lambda)$ is an arbitrary context. We have shown previously that this rule set is absorbing, and it is easy to see that it is equivalent to the rule set consisting of all instances of (N) and (D).

6 Applications: Sequent Calculi for Conditional Logics

After having seen how the construction of absorbing rule sets gives rise to cut-elimination for a number of well-studied normal modal logics, in this section we construct a cut-free sequent calculus for a number of conditional logics.

Conditional logics [2] are extensions of propositional logic by a non-monotonic conditional $A \Rightarrow B$, read as "B holds under the condition that A". The conditional implication is non-monotonic in general, that is the validity of $A \Rightarrow B$ does *not* imply that also $(A \wedge C) \Rightarrow B$ is a valid statement.

Axiomatically, the first argument A of the conditional operation $A \Rightarrow B$ behaves like the $\Box$ in neighbourhood frames and only supports replacement of equivalents, whereas the second argument B obeys the rules of K. We recall from Example 2 that CK CK is axiomatised by the rules (RCEA) and (RCK) that we augment with a subset of the following axioms:

$$(\mathsf{ID})\; A \Rightarrow A \qquad (\mathsf{MP})\; (A \Rightarrow B) \to A \to B \qquad (\mathsf{CEM})\; (A \Rightarrow B) \vee (A \Rightarrow \neg B).$$

The first axiom embodies a form of identity in the sense that A holds under condition A and (MP) is a conditional form of modus ponens. The axiom (CEM) is the conditional excluded middle. We denote combination of rule sets by juxtaposition so that CKID comprises all instances of CK and ID.

6.1 Cut Elimination for Extensions of CK without CEM

We first treat extensions of the basic conditional logic CK with axioms ID and MP, but not including CEM and discuss CEM later, as the effect of adding CEM leads to a more general form of the CK rule. We start by introducing some notation that provides a shorthand for expressing the bi-implications in the premise of CK.

Notation 22. If $A_0, \dots, A_n \in \mathcal{F}(\Lambda)$ are conditional formulas, we write $A_0 = \dots = A_n$ for the sequence of sequents consisting of $\neg A_0, A_i$ and $\neg A_i, A_0$ for all $1 \leq i \leq n$.

If we absorb cuts using Lemmas 19 and 20 we see that all instances of

$$(\mathsf{CK}_g)\frac{A_0 = \dots = A_n \quad \neg B_1, \dots, \neg B_n, B_0}{\neg(A_1 \Rightarrow B_1), \dots, \neg(A_n \Rightarrow B_n), (A_0 \Rightarrow B_0), \Gamma}$$

are admissible in HCK. It is easy to see that the rule set CK_g is actually absorbing:

Theorem 23. *The rule set* CK_g *is absorbing and equivalent to* CK*. As a consequence,* GCK_g *has cut-elimination and* $\mathsf{GCK}_g \vdash A$ *iff* $\mathsf{HCK} \vdash A$ *whenever* $A \in \mathcal{F}(\Lambda)$*.*

Proof. Using Lemmas 19 and Lemma 20 it is immediate that the rule set CK_g is admissible in HCK. The argument that show that CK_g is absorbing is analogous to that for the modal logic K (Example 15), and the result follows from Theorem 18.

The logic CKID arises form CK by adding the identity axiom $A \Rightarrow A$ to the rule set CK_H that axiomatises standard conditional logic. Applying Lemma 19 to the two rule instances on the left

$$\frac{\neg A, B \quad \neg B, A}{\neg(A \Rightarrow A), (A \Rightarrow B)} \qquad A \Rightarrow A \qquad (\mathsf{ID}_g)\frac{A = B}{A \Rightarrow B, \Gamma}$$

gives rise to the rule schema (ID_g) on the right where we have used Lemma 20 to absorb weakening. If we denote the rule set consisting of all instances of CK_g and ID_g by CKID_g, we obtain:

Proposition 24. *The rule set* CKID_g *is absorbing and equivalent to* CKID*.*

Proof. It is easy to see that CKID_g absorbs the structural rules, and that CKID is equivalent to CKID_g. Cuts between conclusions of (ID_g) are readily seen to be absorbed, and absorption of cuts between an instance of CK_g and an instance of ID follows by construction.

The logic CKMP arises by augmenting the logic CK with the additional axiom $(A \Rightarrow B) \to (A \to B)$. The effect of adding (MP) is similar to that of enriching the modal logic K with the (T)-axiom. Adding the missing cuts to CK augmented with (MP) and absorbing the structural rules leads to the rule schema

$$(\mathsf{MP}_g)\frac{A, \neg(A \Rightarrow B), \Gamma \quad \neg B, \neg(A \Rightarrow B), \Gamma}{\neg(A \Rightarrow B), \Gamma}$$

and we denote the rule set consisting of all instances of CK_g and MP_g by CKMP_g. Our cut elimination theorem then takes the following form:

Proposition 25. *The rule set* CKMP_g *is absorbing and equivalent to* CKMP*.*

Proof. Again, it is easy to see that CKMP_g is admissible in HCKMP and the converse follows by construction. All we have to show is that CKMP_g is absorbing, where the absorption of structural rules is easy and left to the reader. For the absorption of cut, the argument is similar to cut elimination in the modal logic T.

6.2 Cut Elimination for Extensions of CKCEM

To construct an absorbing rule set for conditional logic plus the axiom

$$(\mathsf{CEM})(A \Rightarrow B) \vee (A \Rightarrow \neg B)$$

we start from the admissible rule set for CK and close under cuts that arise with (CEM). Repeated applications of Lemma 19 and Lemma 20 lead to the rule set

$$(\mathsf{CKCEM}_g)\frac{A_0 = \cdots = A_n \quad B_0, \ldots, B_j, \neg B_{j+1}, \neg B_n}{(A_0 \Rightarrow B_0), \ldots, (A_j \Rightarrow B_j), \neg(A_{j+1} \Rightarrow B_{j+1}), \ldots, \neg(A_n \Rightarrow B_n), \Gamma}$$

for $1 \leq j \leq n$.

Proposition 26. *The rule set* CKCEM_g *is absorbing and equivalent to* CKCEM.

As a consequence, cut elimination holds in CKCEM_g. We can apply essentially the same argument to an extension of CK with both conditional modus ponens and conditional excluded middle, but have to take care of the cuts arising between MP_g and CKCEM_g, which leads to the new rule

$$(\mathsf{MPEM}_g)\frac{A, (A \Rightarrow B), \Gamma \qquad B, (A \Rightarrow B), \Gamma}{(A \Rightarrow B), \Gamma}.$$

If we denote the extension of CKCEM_g with MP_g and MPEM_g by $\mathsf{CKCEMMP}_g$, we obtain:

Proposition 27. $\mathsf{CKCEMMP}_g$ *is absorbing and equivalent to* $\mathsf{CKCEMMP}$.

We note that the latter theorem was left as an open problem for the sequent system presented in [8]. In summary, we obtain the following results about extensions of the conditional logic CK.

Theorem 28. *Suppose that* L *is one of* CK, CKID, CKMP, CKCEM *or* $\mathsf{CKCEMMP}$. *Then* $\mathsf{GL}_g \vdash A$ *whenever* $\mathsf{HL} \vdash A$ *for all* $A \in \mathcal{F}(\Lambda)$. *Moreover, cut elimination holds in* GL.

The theorem follows, in each of the cases, from Theorem 14 and Theorem 18 together with the fact that the rule set L and L_g are equivalent and the latter is absorbing.

7 Complexity of Proof Search

It is comparatively straightforward to extract complexity bounds for provability of the logics considered above by analysing the complexity of proof search under suitable strategies in the cut-free sequent systems obtained. Clearly, in those cases where all modal rules peel off exactly one layer of modal operators, the depth of proofs is polynomial in the nesting depth of modal operators in the target formula, and therefore, proof search is in PSPACE under mild assumptions on the branching width of proofs [12,9]. Besides reproving Ladner's classical result for K [7], we thus have

Theorem 29. *Provability in* CK *and* CKID *is in PSPACE.*

This reproves known complexity bounds originally shown in [8] (alternative short proofs using coalgebraic semantics are given in [11]). For CKCEM, the bound can be improved using dynamic programming in the same style as in [13]:

Theorem 30. *Provability in* CKCEM *is in coNP.*

More interesting are those cases where some of the modal operators from the conclusion remain in the premise, such as T, $\mathsf{K4}$, CKMP, and CKCEM (where the difference between non-iterative logics, i.e. ones whose Hilbert-axiomatisation does not use nested modalities, such as T, CKMP, and $\mathsf{CKMPCEM}$, and iterative logics such as $\mathsf{K4}$ is surprisingly hard to spot in the sequent presentation). For $\mathsf{K4}$, the standard approach is to consider proofs of minimal depth, which therefore never attempt to prove a sequent repeatedly, and analyse the maximal depth that a branch of a proof can have without repeating a sequent. For T, a different strategy is used, where the (T) rule is limited to be applied at most once to every formula of the form $\neg\Box A$ in between two applications of (K) [6]. A similar strategy works for the conditional logics CKMP and $\mathsf{CKMPCEM}$, which we explain in some additional detail for CKMP.

We let CKMP_g^0 and CKMP_g^1 denote restricted sequent systems where in CKMP_g^0, a formula $\neg(A \Rightarrow B)$ is marked on a branch as soon as the rule (MP_g) has been applied to it (backwards) and unmarked only at the next application of rule (CK_g). Rule (MP_g) applies only to unmarked formulas. In CKMP_g^1, we instead impose a similar restriction where rule (MP_g) applies to a sequent $\neg(A \Rightarrow B), \Gamma$ only in case Γ does not contain a propositional descendant of either A or $\neg B$. Here, a sequent Δ is called a *propositional descendant* of a formula A if it can be generated from A by applying propositional sequent rules backwards (e.g. the propositional descendants of $(\neg(A \wedge B) \wedge C)$ are $\neg(A \wedge B)$; C; and $\neg A, \neg B$). It is easy to check that CKMP_g^1-proofs can be converted into CKMP_g^0-proofs, i.e. CKMP_g^1 is the most restrictive system. One shows that CKMP_g^1 admits contraction and inversion by verifying that the corresponding proof transformations in CKMP_g preserve CKMP_g^1-proofs. It is then clear that every application of the rule (MP_g) that violates the CKMP_g^1-restriction can be replaced by inversion and contraction, so that CKMP_g^1, and hence also CKMP_g^0, proves the same formulas as CKMP_g. Proofs in CKMP_g^0 are easily seen to have at most polynomial depth. Essentially the same reasoning applies to $\mathsf{CKMPCEM}$. Therefore, we have

Theorem 31. *Provability in* CKMP *is in PSPACE; provability in* $\mathsf{CKMPCEM}$ *is in coNP.*

We note that the complexity of $\mathsf{CKMPCEM}$ was explicitly left open in [8].

8 Conclusions

We have established a generic method of cut elimination in modal sequent system based on absorption of cut and structural rules by sets of modal rules. We have applied this method in particular to various conditional logics, thus obtaining cut-free unlabelled sequent calculi that complement recently introduced labelled calculi [8]. In at least one case, the conditional logic $\mathsf{CKMPCEM}$ with modus ponens and conditional excluded middle, our calculus seems to be the first cut-free calculus in the literature, as cut elimination for the corresponding calculus in [8] was explicitly left open. We have applied these calculi to obtain complexity bounds on proof search in conditional logics; in particular we have reproved known upper complexity bounds for CK, CKID, CKMP [8] and improved the bound for CKCEM from PSPACE to coNP using dynamic programming techniques following [13]. Moreover, we have obtained an upper bound coNP for CKMPCEM, for which no bound has previously been published. We conjecture that our general method can also be applied to other base logics, e.g. intuitionistic propositional logic or first-order logic, which is subject to further investigations.

References

1. Avron, A., Lev, I.: Canonical propositional gentzen-type systems. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 529–544. Springer, Heidelberg (2001)
2. Chellas, B.: Modal Logic. Cambridge University Press, Cambridge (1980)
3. Ciabattoni, A., Galatos, N., Terui, K.: From axioms to analytic rules in nonclassical logics. In: Logic in Computer Science, LICS 2008, pp. 229–240. IEEE Press, Los Alamitos (2008)
4. Ciabattoni, A., Terui, K.: Towards a semantic characterization of cut-elimination. Stud. Log. 82, 95–119 (2006)
5. Gentzen, G.: Untersuchungen über das logische Schließen. Math. Z. 39, 176–210 (1934)
6. Heuerding, A., Seyfried, M., Zimmermann, H.: Efficient loop-check for backward proof search in some non-classical propositional logics. In: Miglioli, P., Moscato, U., Ornaghi, M., Mundici, D. (eds.) TABLEAUX 1996. LNCS, vol. 1071, pp. 210–225. Springer, Heidelberg (1996)
7. Ladner, R.E.: The computational complexity of provability in systems of modal propositional logic. SIAM J. Comput. 6 (1977)
8. Olivetti, N., Pozzato, G.L., Schwind, C.: A sequent calculus and a theorem prover for standard conditional logics. ACM Trans. Comput. Logic 8(4) (2007)
9. Pattinson, D., Schröder, L.: Admissibility of cut in coalgebraic logics. In: Coalgebraic Methods in Computer Science, CMCS 2008. ENTCS, vol. 203, pp. 221–241. Elsevier, Amsterdam (2008)
10. Rasga, J.: Sufficient conditions for cut elimination with complexity analysis. Ann. Pure Appl. Logic 149, 81–99 (2007)
11. Schröder, L., Pattinson, D.: Shallow models for non-iterative modal logics. In: Dengel, A.R., Berns, K., Breuel, T.M., Bomarius, F., Roth-Berghofer, T.R. (eds.) KI 2008. LNCS(LNAI), vol. 5243, pp. 324–331. Springer, Heidelberg (2008)
12. Schröder, L., Pattinson, D.: PSPACE bounds for rank-1 modal logics. ACM Trans. Comput. Log. 10(2:13), 1–33 (2009)
13. Vardi, M.: On the complexity of epistemic reasoning. In: Logic in Computer Science, LICS 1989, pp. 243–251. IEEE, Los Alamitos (1989)

Proof Search and Counter-Model Construction for Bi-intuitionistic Propositional Logic with Labelled Sequents

Luís Pinto[1] and Tarmo Uustalu[2]

[1] Centro de Matemática, Universidade do Minho,
Campus de Gualtar, P-4710-057 Braga, Portugal
luis@math.uminho.pt
[2] Institute of Cybernetics at Tallinn University of Technology,
Akadeemia tee 21, EE-12618 Tallinn, Estonia
tarmo@cs.ioc.ee

Abstract. Bi-intuitionistic logic is a conservative extension of intuitionistic logic with a connective dual to implication, called exclusion. We present a sound and complete cut-free labelled sequent calculus for bi-intuitionistic propositional logic, **BiInt**, following S. Negri's general method for devising sequent calculi for normal modal logics. Although it arises as a natural formalization of the Kripke semantics, it is does not directly support proof search. To describe a proof search procedure, we develop a more algorithmic version that also allows for counter-model extraction from a failed proof attempt.

1 Introduction

Bi-intuitionistic logic (also known as Heyting-Brouwer logic, subtractive logic) is an extension of intuitionistic logic with a connective dual to implication, called exclusion (coimplication, subtraction), a symmetrization of intuitionistic logic. It first got the attention of C. Rauszer [14,15,16], who studied its algebraic and Kripke semantics, alongside adequate Hilbert-style systems and sequent calculi. More recently, it has been of interest to Łukowski [9], Restall [17], Crolard [2] and Goré with colleagues [6,1,7,8]. Part of the motivation is the expected computational significance of the logic: one would expect proof systems working as languages for programming with values and continuations in a symmetric way.

A particularity of bi-intuitionistic logic is that it admits simple sequent calculi obtained from the standard ones for intuitionistic logic essentially by dualizing the rule for implication. Although several authors have stated or “proved” that these calculi enjoy cut elimination (most notably Rauszer [15] for her sequent calculus), they are in fact incomplete without cut and thus not directly suitable for backward (i.e., root-first) proof search. The reasons of the failure are similar to those for the modal logic **S5** (**S4** + symmetry) and the future-past tense logic **KtT4** (**S4** + modalities for the converse of the accessibility relation). A closer analysis suggests that finding remedies that are satisfactory, both from

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 295–309, 2009.

the structural proof theory and automated theorem proving points of view, is challenging and provides insights into the subtleties of the logic.

In this paper we propose one solution to the problem. We describe a cut-free labelled sequent calculus for bi-intuitionistic propositional logic, **BiInt**, where the labels are interpreted as worlds in Kripke structures. Exploiting the fact that **BiInt** admits a translation to the future-past tense logic **KtT4**, we obtain it by the general method of S. Negri [12] for devising sequent calculi for normal modal logics. Then, to formulate a search procedure and obtain a termination argument we fine-tune it for the constructive logic situation with monotonicity of truth. This approach is in line with S. Negri's method where frame conditions are uniformly transformed into inference rules, but termination of proof search of the resulting sequent calculus must be obtained on a case-by-case basis. Interestingly, bi-intuitionistic logic turns out to be a rather delicate case.

Cut-free sequent calculi for **BiInt** have also been proposed by Goré and colleagues. Goré's first formulation [6] was in the display logic format, inspired by a general method for devising display systems for normal modal logics. The next formulation by Postniece and Goré [1,7] achieves cut-freedom by combining refutation with proof (passing failure information from premise to premise) to be able to glue counter-models together without the risk of violating the monotonicity condition of interpretations. The new nested sequent calculus by Goré, Postniece and Tiu [8] is a refinement of the display logic version and basically allows reasoning in a local world of a Kripke structure with references to facts about its neighbouring worlds captured in the nested structure.

The paper is organized as follows. In Sect. 2, we introduce **BiInt** with its Kripke semantics and the translation to **KtT4**. We also show its Dragalin-style sequent calculus and why cut elimination fails. In Sect. 3, we introduce a labelled sequent calculus for **BiInt** designed according to S. Negri's recipe. In Sect. 4, we refine this declarative system into a more algorithmic version, show that it is sound and its rules also preserve falsifiability. In the next section (Sect. 5) we define a proof search procedure for the calculus and show that it terminates. In Sect. 6 we put the pieces together to conclude completeness. In the final section we sum up and outline some directions for further enquiry.

2 Bi-intuitionistic Propositional Logic, Dragalin-Style Sequent Calculus and Failure of Cut Elimination

We start by defining the logic **BiInt**. The language extends that of intuitionistic propositional logic, **Int**, by one connective, exclusion, thus the *formulae* are given by the grammar:

$$A, B := p \mid \top \mid \bot \mid A \wedge B \mid A \vee B \mid A \supset B \mid A \prec B$$

where p ranges over a denumerable set of *propositional variables* which give us atoms; the formula $A \prec B$ is the *exclusion* of B from A. We do not take negations as primitive, but in addition to the intuitionistic (or strong) negation, we have dual-intuitionistic (or weak) negation, definable by $\neg A := A \supset \bot$ and $\backsim A := \top \prec A$.

The Kripke semantics defines truth relative to worlds in Kripke structures that are the same as for **Int**. A *Kripke structure* is a triple $K = (W, \leq, I)$ where W is a non-empty set whose elements we think of as *worlds*, $\leq$ is a preorder (reflexive-transitive binary relation) on W (the *accessibility relation*) and I—the *interpretation*—is an assignment of sets of propositional variables to the worlds, which is monotone w.r.t. $\leq$, i.e., whenever $w \leq w'$, we have $I(w) \subseteq I(w')$.

Truth in Kripke structures is defined as for **Int**, but covers also exclusion, interpreted dually to implication as possibility in the past:

- $w \models p$ iff $p \in I(w)$;
- $w \models \top$ always; $w \models \bot$ never;
- $w \models A \wedge B$ iff $w \models A$ and $w \models B$; $w \models A \vee B$ iff $w \models A$ or $w \models B$;
- $w \models B \supset A$ iff, for any $w' \geq w$, $w' \not\models B$ or $w' \models A$;
- $w \models A \prec B$ iff, for some $w' \leq w$, $w' \models A$ and $w' \not\models B$.

A formula is called *valid* if it is true in all worlds of all structures. It is easy to see that monotonicity extends from atoms to all formulae thanks to the universal and existential semantics of implication and exclusion.

It is also a basic observation that the Gödel translation of **Int** into the modal logic **S4** extends to a translation into the future-past tense logic **KtT4** (cf. [9]). As the semantics of **KtT4** does not enforce monotonicity of interpretations, atoms must be translated as future necessities or past possibilities (these are always monotone): $p^{\#} = \Box p$ (or $\blacklozenge p$); $\top^{\#} = \top$; $\bot^{\#} = \bot$; $(A \wedge B)^{\#} = A^{\#} \wedge B^{\#}$; $(A \vee B)^{\#} = A^{\#} \vee B^{\#}$; $(B \supset A)^{\#} = \Box(B^{\#} \supset A^{\#})$; $(A \prec B)^{\#} = \blacklozenge(A^{\#} \prec B^{\#})$.

A sequent calculus for **BiInt** is most easily obtained from Dragalin's sequent calculus for **Int** (as has been done by Restall [17] and Crolard [2]; Rauszer's [15] original sequent calculus was different). In Dragalin's system sequents are multiple-conclusion, but the implication-right rule is constrained. The extension imposes a dual constraint on the exclusion-left rule. The *sequents* are pairs $\Gamma \vdash \Delta$ where Γ, Δ (the *antecedent* and *succedent*) are finite multisets of formulae (we omit braces and denote union by comma as usual). Such a sequent is taken to be *valid* if, for any Kripke structure K and world w, some formula in Γ is false or some formula in Δ is true. The inference rules are displayed in Fig. 1.

Note that the context Δ is missing in the premise of the $\supset R$ rule and dually in the premise of $\prec L$ we do not have the context Γ. The rules $\supset L$ and $\prec R$ involve some contraction. This is necessary because we have chosen not to include a general contraction rule.

This calculus is sound and complete w.r.t. the above-defined notion of validity (completeness can be shown going through the algebraic semantics in terms of Heyting-Brouwer algebras [14]). However it is incomplete without cut, as shown by Pinto and Uustalu in 2003 (private email message from T. Uustalu to R. Goré, 13 Sept. 2004, quoted in [1]). It suffices to consider the obviously valid sequent

initial rule and cut:

$$\frac{}{\varGamma, A \vdash A, \varDelta}\, \mathit{hyp} \qquad \frac{\varGamma \vdash A, \varDelta \quad \varGamma, A \vdash \varDelta}{\varGamma \vdash \varDelta}\, \mathit{cut}$$

logical rules:

$$\frac{\varGamma \vdash \varDelta}{\varGamma, \top \vdash \varDelta}\, \top L \qquad \frac{}{\varGamma \vdash \top, \varDelta}\, \top R \qquad \frac{\varGamma, A, B \vdash \varDelta}{\varGamma, A \wedge B \vdash \varDelta}\, \wedge L \qquad \frac{\varGamma \vdash A, \varDelta \quad \varGamma \vdash B, \varDelta}{\varGamma \vdash A \wedge B, \varDelta}\, \wedge R$$

$$\frac{}{\varGamma, \bot \vdash \varDelta}\, \bot L \qquad \frac{\varGamma \vdash \varDelta}{\varGamma \vdash \bot, \varDelta}\, \bot R \qquad \frac{\varGamma, A \vdash \varDelta \quad \varGamma, B \vdash \varDelta}{\varGamma, A \vee B \vdash \varDelta}\, \vee L \qquad \frac{\varGamma \vdash A, B, \varDelta}{\varGamma \vdash A \vee B, \varDelta}\, \vee R$$

$$\frac{\varGamma, B \supset A \vdash B, \varDelta \quad \varGamma, A \vdash \varDelta}{\varGamma, B \supset A \vdash \varDelta}\, \supset L \qquad \frac{\varGamma, B \vdash A}{\varGamma \vdash B \supset A, \varDelta}\, \supset R$$

$$\frac{A \vdash B, \varDelta}{\varGamma, A \prec B \vdash \varDelta}\, \prec L \qquad \frac{\varGamma \vdash A, \varDelta \quad \varGamma, B \vdash A \prec B, \varDelta}{\varGamma \vdash A \prec B, \varDelta}\, \prec R$$

Fig. 1. Dragalin-style sequent calculus for **BiInt**

$p \vdash q, r \supset ((p \prec q) \wedge r)$. The only possible last inference in a proof could be

$$\frac{\overset{?}{p, r \vdash (p \prec q) \wedge r}}{p \vdash q, r \supset ((p \prec q) \wedge r)}\, \supset R$$

but the premise is invalid as the succedent formula q has been lost. With cut, the sequent is proved as follows:

$$\frac{\dfrac{\dfrac{}{p \vdash q, p, \ldots}\, \mathit{hyp} \quad \dfrac{}{p, q \vdash q, p \prec q, \ldots}\, \mathit{hyp}}{p \vdash q, p \prec q, \ldots}\, \prec R \qquad \dfrac{\dfrac{\dfrac{}{p, p \prec q, r \vdash p \prec q}\, \mathit{hyp} \quad \dfrac{}{p, p \prec q, r \vdash r}\, \mathit{hyp}}{p, p \prec q, r \vdash (p \prec q) \wedge r}\, \wedge R}{p, p \prec q \vdash q, r \supset ((p \prec q) \wedge r)}\, \supset R}{p \vdash q, r \supset ((p \prec q) \wedge r)}\, \mathit{cut}$$

Cut elimination fails as we cannot permute the cut on the exclusion $p \prec q$ up past the $\supset R$ inference for which the cut formula is a side formula. This is one type of cuts that cannot be eliminated, there are altogether 3 such types [11]. This situation is similar to the naive sequent calculus for **S5** where the sequent $p \vdash \Box \Diamond p$ cannot be proved without cut, but can be proved by applying cut to the sequents $p \vdash \Diamond p$ and $\Diamond p \vdash \Box \Diamond p$ that are provable without cut.

3 L: A Labelled Sequent Calculus

We now proceed to a labelled sequent calculus for bi-intuitionistic logic that we call **L**. This calculus turns out to be complete without a cut rule. Essentially it is a formalization of the first-order theory of the Kripke semantics in such a fashion that the extralogical axioms corresponding to the reflexivity-transitivity condition on frames and monotonicity condition on interpretations do not necessitate cut. Our design follows the method of S. Negri [12].

We proceed from a denumerable set of *labels*. A *labelled formula* is a pair $x : A$ where x is a label and A a formula. The intended meaning is truth of the formula at a particular world.

preorder rules:

$$\frac{\Gamma \vdash_{G\cup\{(x,x)\}} \Delta}{\Gamma \vdash_G \Delta}\,\mathit{refl} \qquad \frac{xGy \quad yGz \quad \Gamma \vdash_{G\cup\{(x,z)\}} \Delta}{\Gamma \vdash_G \Delta}\,\mathit{trans}$$

initial rule and monotonicity rules:

$$\frac{}{\Gamma, x : A \vdash_G x : A, \Delta}\,\mathit{hyp} \qquad \frac{xGy \quad \Gamma, x : A, y : A \vdash_G \Delta}{\Gamma, x : A \vdash_G \Delta}\,\mathit{monL} \qquad \frac{yGx \quad \Gamma \vdash_G y : A, x : A, \Delta}{\Gamma \vdash_G x : A, \Delta}\,\mathit{monR}$$

logical rules:

$$\frac{\Gamma \vdash_G \Delta}{\Gamma, x : \top \vdash_G \Delta}\,\top L \qquad \frac{}{\Gamma \vdash_G x : \top, \Delta}\,\top R \qquad \frac{\Gamma, x : A, x : B \vdash_G \Delta}{\Gamma, x : A \wedge B \vdash_G \Delta}\,\wedge L \qquad \frac{\Gamma \vdash_G x : A, \Delta \quad \Gamma \vdash_G x : B, \Delta}{\Gamma \vdash_G x : A \wedge B, \Delta}\,\wedge R$$

$$\frac{}{\Gamma, x : \bot \vdash_G \Delta}\,\bot L \qquad \frac{\Gamma \vdash_G \Delta}{\Gamma \vdash_G x : \bot, \Delta}\,\bot R \qquad \frac{\Gamma, x : A \vdash_G \Delta \quad \Gamma, x : B \vdash_G \Delta}{\Gamma, x : A \vee B \vdash_G \Delta}\,\vee L \qquad \frac{\Gamma \vdash_G x : A, x : B, \Delta}{\Gamma \vdash_G x : A \vee B, \Delta}\,\vee R$$

$$\frac{xGy \quad \Gamma \vdash_G y : B, \Delta \quad \Gamma, y : A \vdash_G \Delta}{\Gamma, x : B \supset A \vdash_G \Delta}\,\supset L \qquad \frac{y \notin G, \Gamma, \Delta \quad \Gamma, y : B \vdash_{G\cup\{(x,y)\}} y : A, \Delta}{\Gamma \vdash_G x : B \supset A, \Delta}\,\supset R$$

$$\frac{y \notin G, \Gamma, \Delta \quad \Gamma, y : A \vdash_{G\cup\{(y,x)\}} y : B, \Delta}{\Gamma, x : A \prec B \vdash_G \Delta}\,\prec L \qquad \frac{yGx \quad \Gamma \vdash_G y : A, \Delta \quad \Gamma, y : B \vdash_G \Delta}{\Gamma \vdash_G x : A \prec B, \Delta}\,\prec R$$

Fig. 2. Labelled sequent calculus **L**

Sequents are triples $\Gamma \vdash_G \Delta$ where Γ and Δ are finite multisets of labelled formulae, and G is a finite binary relation on labels called the *graph*. Graphs are a means to keep track of label dependencies and thus induce an accessibility relation on worlds.

The inference rules are presented in Fig. 2. Some of them have provisos, that we also write as rule premises. We let xGy abbreviate $(x, y) \in G$. Following usual sequent calculus terminology, at a given rule, we call the explicit labelled formula in the conclusion the *labelled formula introduced* by the rule or the *main labelled formula* of the rule and the explicit labelled formulae in the premises the *side labelled formulae.*

The interesting logical rules are those for implication and exclusion which are dual. Notice the freshness condition on the label y in the rules $\supset R$ and $\prec L$, guaranteeing their soundness. We call label y the *eigenlabel* of the rule and x the *parent* of y. Note also the presence of the monotonicity rules accounting for propagation of truth (resp. falsity) to future (resp. past) worlds and preorder rules which account for reflexivity and transitivity of accessibility.

The counter-example to cut elimination for the Dragalin-style sequent calculus is proved in **L** as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : p, y : r \vdash_{(x,y)} x : q, x : p}\,\mathit{hyp} \quad \dfrac{}{x : p, y : r, x : q \vdash_{(x,y)} x : q}\,\mathit{hyp}}{x : p, y : r \vdash_{(x,y)} x : q, y : p \prec q}\,\prec R \quad \dfrac{}{x : p, y : r \vdash_{(x,y)} x : q, y : r}\,\mathit{hyp}}{x : p, y : r \vdash_{(x,y)} x : q, y : (p \prec q) \wedge r}\,\wedge R}{x : p \vdash_{\emptyset} x : q, x : r \supset ((p \prec q) \wedge r)}\,\supset R$$

Notice the downward information propagation in the $\prec R$ inference to an already existing label.

In a **L**-derivation the names of the eigenlabels can be changed (to new names not occurring in the derivation) without changing the end sequent. This property allows us to show by usual methods that **L** enjoys admissibility of the weakening rules. A simple combination of the monotonicity, reflexivity and weakening also guarantees admissibility of the contraction rules in **L**. (This is what enables us to avoid explicit contractions at $\supset L$ and $\prec R$ rules.)

The cut rule is also admissible in **L**. This can be proved along the lines of cut elimination results of S. Negri for labelled sequent calculi for modal logics. In this paper, as an immediate consequence of soundness and completeness of system **L** w.r.t. the Kripke semantics (Cor. 1), we get a semantical proof of admissibility of cut.

Given a Kripke structure K, a K*-valuation* is a mapping from the set of labels to the set of worlds of K.

Definition 1. *A Kripke structure* $K = (W, \leq, I)$ *and a* K*-valuation* v *are a* counter-model *(cm) to an* **L***-sequent* $\Gamma \vdash_G \Delta$*, if: i) for all* xGy*,* $v(x) \leq v(y)$*; ii) for all* $x : A \in \Gamma$*,* $v(x) \models A$*; and iii) for all* $x : A \in \Delta$*,* $v(x) \not\models A$*. The sequent is* valid*, if it has no counter-model.*

Proposition 1 (Soundness of L). *If* $\Gamma \vdash_G \Delta$ *is derivable,* $\Gamma \vdash_G \Delta$ *is valid.*

Completeness holds as well (Cor. 1) and is proved with the help of the algorithmic version of **L** introduced in the next section. In fact our completeness argument allows for construction of counter-models of non-derivable sequents.

4 L*: An Algorithmic Version of L

Although **L** constitutes a good basis for backward proof search for bi-intuitionistic propositional logic, it still faces the problem that the preorder and monotonicity rules can be applied at any point in backward proof search. To deal with this problem, we introduce now an algorithmic version of **L** called $\mathbf{L}^*$. System $\mathbf{L}^*$ does not have explicit preorder or monotonicity rules. It uses a marking mechanism on certain kinds of labelled formulae. Such mechanism allows for the recovering of labelled formulae, so that monotonicity requirements are guaranteed. The marking mechanism is also designed in a way that it can be used in loop-detection, to avoid infinite search along paths corresponding to non-derivable sequents.

Sequents in $\mathbf{L}^*$ are triples $\Gamma \vdash_G \Delta$ as in **L**, with the difference that, in the contexts Γ and Δ, labelled formulae can now be marked either with $*$ (written as $x : A^*$), $\circ$ (written as $x : A^\circ$) or with $\bullet$ (written as $x : A^\bullet$). The rules of $\mathbf{L}^*$ are in Fig. 3.

Let us briefly explain the role of $^+$ and $-$ and of marks $*$, $\circ$ and $\bullet$ in backward proof search. The $^+$ (resp. $-$) is used to propagate a formula to future (resp. past) labels (as determined by the transitive closure of the graph). The marking

initial rule:

$$\frac{}{\Gamma, x : p^{\circ} \vdash_G x : p^{\circ}, \Delta}\ hyp$$

atom rules:

$$\frac{\Gamma, p^{+}, x : p^{*}, x : p^{\circ} \vdash_G \Delta}{\Gamma, x : p \vdash_G \Delta}\ atomL \qquad \frac{\Gamma \vdash_G x : p^{\circ}, x : p^{*}, p^{-}, \Delta}{\Gamma \vdash_G x : p, \Delta}\ atomR$$

$$\text{where } p^{+} = \{y : p \mid xGy\} \qquad\qquad \text{where } p^{-} = \{y : p \mid yGx\}$$

logical rules:

$$\frac{\Gamma \vdash_G \Delta}{\Gamma, x : \top \vdash_G \Delta}\ \top L \quad \frac{}{\Gamma \vdash_G x : \top, \Delta}\ \top R \quad \frac{\Gamma, x : A, x : B \vdash_G \Delta}{\Gamma, x : A \wedge B \vdash_G \Delta}\ \wedge L \quad \frac{\Gamma \vdash_G x : A, \Delta \quad \Gamma \vdash_G x : B, \Delta}{\Gamma \vdash_G x : A \wedge B, \Delta}\ \wedge R$$

$$\frac{}{\Gamma, x : \bot \vdash_G \Delta}\ \bot L \quad \frac{\Gamma \vdash_G \Delta}{\Gamma \vdash_G x : \bot, \Delta}\ \bot R \quad \frac{\Gamma, x : A \vdash_G \Delta \quad \Gamma, x : B \vdash_G \Delta}{\Gamma, x : A \vee B \vdash_G \Delta}\ \vee L \quad \frac{\Gamma \vdash_G x : A, x : B, \Delta}{\Gamma \vdash_G x : A \vee B, \Delta}\ \vee R$$

$$\frac{\Gamma, (B \supset A)^{+}, x : (B \supset A)^{*} \vdash_G x : B, \Delta \quad \Gamma, x : A \vdash_G \Delta}{\Gamma, x : B \supset A \vdash_G \Delta}\ \supset L$$

$$\text{where } (B \supset A)^{+} = \{y : B \supset A \mid xGy\}$$

$$\frac{x : (B \supset A)^{\bullet} \notin \Delta \quad y \notin G, \Gamma, \Delta, \quad \Gamma, \Gamma^{y/x}, y : B \vdash_{G \cup \{(x,y)\}} y : A, x : (B \supset A)^{\bullet}, \Delta}{\Gamma \vdash_G x : B \supset A, \Delta}\ \supset R$$

$$\begin{aligned}\text{where } \Gamma^{y/x} = {} & \{y : C \mid x : C^{*} \in \Gamma\} \cup \{y : p^{\circ} \mid x : p^{\circ} \in \Gamma\} \\ & \cup \{y : (C \prec D)^{\bullet} \mid x : C \prec D \in \Gamma \text{ or } x : (C \prec D)^{\bullet} \in \Gamma\}\end{aligned}$$

$$\frac{x : (A \prec B)^{\bullet} \notin \Gamma \quad y \notin G, \Gamma, \Delta \quad \Gamma, x : (A \prec B)^{\bullet}, y : A \vdash_{G \cup \{(y,x)\}} y : B, \Delta^{y/x}, \Delta}{\Gamma, x : A \prec B \vdash_G \Delta}\ \prec L$$

$$\begin{aligned}\text{where } \Delta^{y/x} = {} & \{y : C \mid x : C^{*} \in \Delta\} \cup \{y : p^{\circ} \mid x : p^{\circ} \in \Delta\} \\ & \cup \{y : (D \supset C)^{\bullet} \mid x : D \supset C \in \Delta \text{ or } x : (D \supset C)^{\bullet} \in \Delta\}\end{aligned}$$

$$\frac{\Gamma \vdash_G x : A, \Delta \quad \Gamma, x : B \vdash_G x : (A \prec B)^{*}, (A \prec B)^{-}, \Delta}{\Gamma \vdash_G x : A \prec B, \Delta}\ \prec R$$

$$\text{where } (A \prec B)^{-} = \{y : A \prec B \mid yGx\}$$

Fig. 3. Algorithmic version $\mathbf{L}^{*}$

$x : A^{*}$ is done at the atom rules, $\supset L$ and $\prec R$ (where $x : A$ is the main formula) in order to be able to recover A at eventual labels still unknown when $x : A$ is analysed, but later created with a graph connection to x. The marking of a labelled formula with a $\circ$ (used only with atoms) or $\bullet$ (used only with implications and exclusions) means essentially that the formula was already analysed (the case with the explicit circles and bullets in the rule premises of the atom rules, $\supset R$, $\prec L$) or has no further useful information and so need not be analysed (the case with circles and bullets implicit in $\Gamma^{y/x}$ and $\Delta^{y/x}$ in the premises of $\supset R$ and $\prec L$ respectively) and prevents a new analysis of the formula at the given world (notice that no rule introduces a labelled formula with a circle or a bullet).

Notice that an $\mathbf{L}$-sequent is also an $\mathbf{L}^{*}$-sequent and that if we take an $\mathbf{L}^{*}$-sequent and erase all $*$, $\circ$ and $\bullet$ marks we obtain an $\mathbf{L}$-sequent. Given an $\mathbf{L}^{*}$

context Γ, we write Γ^- for the **L**-context resulting from it by replacing all labelled formulae $x : A^*$, $x : A^\circ$ with unmarked labelled formulae $x : A$ and removing all labelled formulae $x : A^\bullet$. Given an $\mathbf{L}^*$-sequent $\Gamma \vdash_G \Delta$ its *erasure* is the **L**-sequent $\Gamma^- \vdash_G \Delta^-$. We say that an $\mathbf{L}^*$-rule is derivable in **L** if the rule obtained by replacing its premises and conclusion by their erasures is derivable in **L**. The next proposition shows that all $\mathbf{L}^*$-rules are derivable in **L** and thus $\mathbf{L}^*$ is sound w.r.t. **L**.

Proposition 2 (Soundness of $\mathbf{L}^*$ w.r.t. L). *1. All rules of* $\mathbf{L}^*$ *are derivable in* **L**. *2. If* $\Gamma \vdash_G \Delta$ *is derivable in* $\mathbf{L}^*$ *then* $\Gamma^- \vdash_G \Delta^-$ *is derivable in* **L**.

Because the rules of $\mathbf{L}^*$ do not throw away any relevant information (read backward, i.e., from the conclusion to the premises), they have the strong property that a counter-model of a premise is also a counter-model of the conclusion. This is used in Sec. 6 for extracting counter-models out of failed proof attempts.

Given a Kripke structure $K = (W, \leq, I)$, a K-valuation v and a graph G, $\leq_G^-$ denotes the relation $\leq \backslash v(G)^*$, i.e., $\leq_G^-$ is the relation obtained from $\leq$ by eliminating all pairs in the reflexive-transitive closure of $\{(v(x), v(y)) \mid xGy\}$.

Definition 2. *A Kripke structure* $K = (W, \leq, I)$ *and a* K*-valuation* v *are a* counter-model *of an* $\mathbf{L}^*$*-sequent* $\Gamma \vdash_G \Delta$ *when:*

1. *for all* xGy, $v(x) \leq v(y)$;
2. *for all* $x : A, x : A^\circ \in \Gamma$, $v(x) \models A$;
3. *for all* $x : A^* \in \Gamma$ *and for all* $w \in W$ *such that* $v(x) \leq_G^- w$, $w \models A$;
4. *for all* $x : A, x : A^\circ \in \Delta$, $v(x) \not\models A$;
5. *for all* $x : A^* \in \Delta$ *and for all* $w \in W$ *such that* $w \leq_G^- v(x)$, $w \not\models A$.

Notice that for **L**-sequents this notion of counter-model coincides with the notion introduced in the previous section. As usual *valid sequents* are those for which there are no counter-models.

Proposition 3 (Preservation of counter-models). *For each* $\mathbf{L}^*$*-rule, a counter-model of a premise is also a counter-model of the conclusion.*

5 A Search Procedure and Its Termination

We now describe a backward search procedure for $\mathbf{L}^*$, which incorporates a loop-checking mechanism, and prove it sound and terminating. As a by-product of the explicit presence in sequents of labels/worlds and the graph/accessibility relation, when the search procedure terminates with failure, we will be left with a Kripke counter-model of the given sequent. This fact is proved in the next section and accounts for the completeness of the search procedure. In order to describe the search procedure, we introduce first some terminology, notation and also the loop-rules.

The rules $\supset R$ and $\prec L$ are the only rules of $\mathbf{L}^*$ where the graph relation varies in a backward reading. We call these rules *world creating rules.* A sequent

$$\frac{y \notin G \quad \Gamma \setminus \Gamma(y) \vdash_G \Delta[x/y]}{\Gamma \vdash_{G \cup \{(x,y)\}} \Delta}\ \mathit{loopUp} \qquad \frac{y \notin G \quad \Gamma[x/y] \vdash_G \Delta \setminus \Delta(y)}{\Gamma \vdash_{G \cup \{(y,x)\}} \Delta}\ \mathit{loopDn}$$

provided $\Gamma[y] \subseteq \Gamma[x] \cup \Gamma^{\bullet}[x], \Gamma^{*}[y] \subseteq \Gamma^{*}[x]$, and $\Gamma^{\circ}[y] \subseteq \Gamma^{\circ}[x]$ provided $\Delta[y] \subseteq \Delta[x] \cup \Delta^{\bullet}[x], \Delta^{*}[y] \subseteq \Delta^{*}[x]$, and $\Delta^{\circ}[y] \subseteq \Delta^{\circ}[x]$

Fig. 4. Loop rules

$\Gamma \vdash_G \Delta$ is called *saturated* if it is irreducible w.r.t. the non-world creating rules. A sequent $\Gamma \vdash_G \Delta$ is called *stuck* when it is irreducible w.r.t. any rule and moreover it is not an *axiom* (*hyp*, $\bot L$, $\top R$).

Given an $\mathbf{L}^*$ context Γ, we use the notations $\Gamma[x]$, $\Gamma^*[x]$, $\Gamma^{\circ}[x]$, $\Gamma^{\bullet}[x]$ and $\Gamma(x)$ to mean $\{A \mid x{:}A \in \Gamma\}$, $\{A \mid x{:}A^* \in \Gamma\}$, $\{A \mid x{:}A^{\circ} \in \Gamma\}$, $\{A \mid x{:}A^{\bullet} \in \Gamma\}$ and $\Gamma[x] \cup \Gamma^*[x] \cup \Gamma^{\circ}[x] \cup \Gamma^{\bullet}[x]$ respectively.

The *loop rules* are presented in Fig. 4. (For a context Γ and labels x and y, the notation $\Gamma[x/y]$ stands for the context obtained by replacing y with x in Γ.) Their backward reading corresponds to the action taken when a loop is detected and the detection of a loop corresponds to satisfaction of their side-conditions. The formulation of the loop rules corresponds to the situation where x is the parent and thus y is a descendant of x, labeling necessarily subformulae of x-labelled formulae.

We are now in conditions of presenting the *search procedure.* It goes as follows:

1. Given an $\mathbf{L}^*$-sequent $\Gamma \vdash_G \Delta$, we reduce it w.r.t. the non-world creating rules (i.e., we apply as long as possible these rules). We call a *saturation* both this process and the partial proof of $\Gamma \vdash_G \Delta$ so constructed. The top sequent of each branch of a saturation is a *saturated sequent.* Notice also that the order in which rules are applied in saturation is unimportant since they are inter-permutable.
2. Then, for every branch in the saturation of $\Gamma \vdash_G \Delta$, we do the following:
   (a) we check if the top sequent is an axiom and if so search along the branch is stopped with *success*;
   (b) we check if there is a loop, i.e., we test if the side condition of any of the loop rules is met, and if so proceed according to the corresponding loop rule.
3. If neither (a) nor (b) is the case, the development of the branch carries on, by applying one of the world creating rules, and we go back to 1.We stop with *failure* if no world creating rule can be applied.

We call *proof attempt* both the run of the search procedure with a given sequent and the corresponding partial proof (in $\mathbf{L}^*$ augmented with the loop rules). Throughout we assume that proof attempts always start with $\mathbf{L}$-sequents whose graphs are trees (i.e., the graph, seen as an undirected graph by forgetting the directions of the arcs, is connected and acyclic). Then the graphs of all sequent in the proof attempt are trees.

Proposition 4 (Soundness of the search procedure). *If the proof attempt for an* $\mathbf{L}$*-sequent terminates with success, then the sequent is* $\mathbf{L}$*-derivable.*

Proof: By induction on the height of the proof attempt, we prove that, for any $\mathbf{L}^*$-sequent $\Gamma \vdash_G \Delta$ in it, $\Gamma^- \vdash_G \Delta^-$ is derivable in **L**. The cases corresponding to $\mathbf{L}^*$-inferences follow by part 1. of Prop. 2. Consider the case corresponding to the *loopUp* rule of Fig. 4. (The case of *loopDn* is similar.) By IH we have that $(\Gamma \setminus \Gamma(y))^- \vdash_G \Delta[x/y]^-$ is **L**-derivable. From this, by weakening , $\Gamma^- \vdash_{G\cup\{(x,y)\}} (\Delta \setminus \Delta(y))^-, \Delta(y)[x/y]^-, \Delta(y)^-$is also derivable in **L**. Since xGy, by repeated use of *monR*, we can derive $\Gamma^- \vdash_{G\cup\{(x,y)\}} (\Delta \setminus \Delta(y))^-, \Delta(y)^-$which is $\Gamma^- \vdash_{G\cup\{(x,y)\}} \Delta^-$. □

Now we consider terminology, notation and lemmata used in particular for proving termination of the search procedure. Given a label x, the *world creation tree* of x induced by a branch of a proof attempt has x as root and has as subtrees (if any) the world creation trees of each eigenlabel in the branch whose parent is x. Given a set of formulas S, $\mathsf{mhf}(S)$ stands for the maximal height of the formulae in S.

Lemma 1. *Any saturation in a proof attempt is finite.*

Proof: Observe that: (i) $\wedge$ and $\vee$ inferences replace the main formula by strict subformulae; and (ii) even if the main formula of an *atomL*, *atomR*, $\supset L$ or $\prec R$ inference may reappear in the premises or upper sequents, it does so with a distinct label (as the graph is a tree) and thus can only reappear finitely many times (recall $\mathbf{L}^*$-graphs are finite). □

Lemma 2. *Given a label x and a branch $\mathcal{B}$ of a proof attempt, x has finitely many children in $\mathcal{B}$.*

Proof: Notice that all formulae in a sequent of $\mathcal{B}$ are subformulae of a formula in the end sequent of $\mathcal{B}$ (which is finite) and that, once $x : A \supset B$ (resp. $x : A \prec B$) is analysed as the main formula of a $\supset R$ (resp. $\prec L$) inference, $x : (A \supset B)^\bullet$ (resp. $x : (A \prec B)^\bullet$) is added to the succedent (resp. antecedent) of the inference's premise, preventing that $x : A \supset B$ (resp. $x : A \prec B$) becomes analysed again. □

Lemma 3. *For $\diamond \in \{*, \circ\}$ and any saturated sequent $\Gamma \vdash_{G\cup\{(x,y)\}} \Delta$ in a proof attempt: i) if $x : A^\diamond \in \Gamma$, $y : A^\diamond \in \Gamma$; and ii) if $y : A^\diamond \in \Delta$, $x : A^\diamond \in \Delta$.*

Proof: Firstly notice that, for any $\mathbf{L}^*$-rule, if $z : B^\diamond$ is in the antecedent (resp. succedent) of the conclusion, $z : B^\diamond$ is in the antecedent (resp. succedent) of any premise. Consider the case $x : p^\circ \in \Gamma$ (the other cases being similar or simpler). Then $x : p^* \in \Gamma$ and thus $x : p$ must have been the main formula in an *atomL* inference. Let $\Gamma_0 \vdash_{G_0} \Delta_0$ be the premise of that inference. If $(x, y) \in G_0$, $y : p \in \Gamma_0$ and any top sequent in the saturation of $\Gamma_0 \vdash_{G_0} \Delta_0$ has both $y : p^*$ and $y : p^\circ$ in the antecedent. If not, above the referred inference, there must be an $\supset R$ inference with eigenlabel y and parent x and the top sequents of the saturation of its premise have $y : p^*$ and $y : p^\circ$ in their antecedents. □

Lemma 4. *In a proof attempt, if $\Gamma_0 \vdash_G \Delta_0$ is the conclusion of an $\supset R$ inference with eigenlabel x_1 and parent x_0 and $\Gamma_1 \vdash_{G\cup\{(x_0,x_1)\}} \Delta_1$ is a top sequent in the saturation of the inference's premise, then $\Gamma_0(x_0) \subset \Gamma_1(x_1)$.*

Proof: By the following three facts: i) $\Gamma_0(x_0) \subseteq \Gamma_1(x_0)$, because no $\mathbf{L}^*$ rule removes starred, circled or bulleted formulae (when read backwards); ii) $\Gamma_1(x_0) \subseteq \Gamma_1(x_1)$, because $\Gamma_1[x_0] \cup \Gamma_1^{\bullet}[x_0] \subseteq \Gamma_1^{\bullet}[x_1]$, $\Gamma_1^{*}[x_0] \subseteq \Gamma_1^{*}[x_1]$, $\Gamma_1^{\circ}[x_0] \subseteq \Gamma_1^{\circ}[x_1]$ (the last two containments proved with the help of Lemma 3); iii) $\Gamma_1(x_1) \not\subseteq \Gamma_1(x_0)$, because of the loop checking mechanism. □

Lemma 5. *For any sub-branch of a proof attempt of the form*

$$\begin{array}{c}
\Gamma_2 \vdash_{G_1 \cup \{(x_2,x_1)\}} \Delta_2 \\
\vdots \\
\dfrac{\Gamma_1, x_1 : (A_1 \prec B_1)^{\bullet}, x_2 : A_1 \vdash_{G_1 \cup \{(x_2,x_1)\}} x_2 : B_1, \Delta_1, \Delta_1^{x_2/x_1}}{\Gamma_1, x_1 : A_1 \prec B_1 \vdash_{G_1} \Delta_1} \prec L \\
\vdots \\
\dfrac{\Gamma_0, \Gamma_0^{x_1/x_0}, x_1 : A_0 \vdash_{G_0 \cup \{(x_0,x_1)\}} x_1 : B_0, x_0 : (A_0 \supset B_0)^{\bullet}, \Delta_0}{\Gamma_0 \vdash_{G_0} x_0 : A_0 \supset B_0, \Delta_0} \supset R
\end{array}$$

where the conclusion of $\prec L$ *is a top sequent in the saturation of the premise of* $\supset R$ *and* $\Gamma_2 \vdash_{G_1 \cup \{(x_2,x_1)\}} \Delta_2$ *is a top sequent in the saturation of the premise of* $\prec L$ *we have* $\mathsf{mhf}(\Gamma_0(x_0) \cup \{A_0 \supset B_0\}) > \mathsf{mhf}(\Gamma_2(x_2) \cup \Delta_2(x_2))$.

Proof: By the following two facts: i) each formula of $\Gamma_2(x_2) \cup \Delta_2(x_2)$ is a subformula of $\Delta_1(x_1)$ or a strict subformula of $A_1 \prec B_1$; and ii) $\Delta_1(x_1)$ has only strict subformulae of $\Gamma_0(x_0) \cup \{A_0 \supset B_0\}$ and $A_1 \prec B_1$ is a subformula of $\Gamma_0(x_0) \cup \{A_0 \supset B_0\}$[1]. □

Theorem 1 (Termination of the search procedure). *A proof attempt always terminates.*

Proof: If it did not, by König's Lemma there would be an infinite branch $\mathcal{B}$ in the proof attempt. Since each saturation is finite (Lemma 1), there must be infinitely many saturations and at least one of the labels in the end sequent, x_{00} say, has an infinite world creation tree, call it $\mathcal{T}$. By Lemma 2, $\mathcal{T}$ is finitely branching and so König's Lemma forces $\mathcal{T}$ to have an infinite branch, which we will show to be impossible.

[1] The first fact follows from the following lemma (and the second fact from an analogous lemma): For any sequent $\Gamma \vdash_{G_1 \cup \{(x_2,x_1)\}} \Delta$ in the saturation of the premise of an $\prec L$ inference with eigenlabel x_2 and parent x_1:

1. if $x : A \in \Gamma$ (resp. Δ) and A is not an exclusion (resp. implication), then $x_2(G_1 \cup \{(x_2,x_1)\})^* x$ (resp. $x(G_1 \cup \{(x_2,x_1)\})^* x_2$);
2. if $A \in \Gamma[x_1]$ (resp. $\Delta[x_1]$), then A is an exclusion (resp. implication) or $A \in \Gamma(x_2)$ (resp. $\Delta(x_2)$);
3. if $\Gamma_0 \vdash_{G_1 \cup \{(x_2,x_1)\}} \Delta_0$ is a sequent immediately above $\Gamma \vdash_{G_1 \cup \{(x_2,x_1)\}} \Delta$ and $A \in \Gamma_0(x_2)$ (resp. $\Delta_0(x_2)$), then A is a subformula of $\Gamma(x_2)$ (resp. $\Delta(x_2)$) or a strict subformula of $\Delta(x_2)$ (resp. $\Gamma(x_2)$).

It is impossible that an infinite branch of $\mathcal{T}$ beyond a certain point, z_0 say, goes always upwards, i.e., that the descendants $z_1, z_2, \ldots$ of z_0 in $\mathcal{T}$ all arise with $\supset R$ inferences. Otherwise, using Lemma 4, we could form an infinite sequence

$$\Gamma_0(z_0) \ \subset \ \Gamma_1(z_1) \ \subset \ \Gamma_2(z_2) \ \ldots$$

(where Γ_i is the conclusion's antecedent of the inference where z_i creates z_{i+1}), which is is impossible, for all these sets must be included in the finite set of subformulae of the end sequent. Similarly, $\mathcal{T}$ cannot have an infinite branch that beyond a certain point goes always downwards.

Therefore, an infinite branch of $\mathcal{T}$ would have to correspond to an infinite zigzag of the shape shown in Fig. 5 (or of a dual shape), where a dashed arrow up (resp. down) means that zero or more worlds were created by $\supset R$ (resp. $\prec L$) inferences in between x_{i0} and x_{in_i} and a solid arrow up (resp. down) means that $x_{(i+1)0}$ was created from x_{in_i} by an $\supset R$ (resp. $\prec L$) inference. Let Γ_{ij} (resp. Δ_{ij}) stand for the conclusion's antecedent (resp. succedent) of the inference where x_{ij} creates its immediate successor in branch $\mathcal{B}$. By Lemma 5, a property analogous to it (for the case where $\prec L$ is below $\supset R$), and the fact that, given i and $j_1 < j_2 \leq n_i$, $\mathsf{mhf}(\Gamma_{ij_1}(x_{ij_1}) \cup \Delta_{ij_1}(x_{ij_1})) \geq \mathsf{mhf}(\Gamma_{ij_2}(x_{ij_2}) \cup \Delta_{ij_2}(x_{ij_2}))$, it follows that $\mathsf{mhf}(\Gamma_{in_i}(x_{in_i}) \cup \Delta_{in_i}(x_{in_i})) \ > \ \mathsf{mhf}(\Gamma_{(i+1)n_{i+1}}(x_{(i+1)n_{i+1}}) \cup \Delta_{(i+1)n_{i+1}}(x_{(i+1)n_{i+1}}))$ and so an infinite descending chain of natural numbers would be produced. □

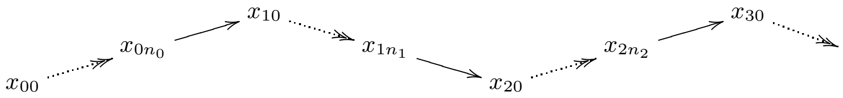


Fig. 5. An infinite zigzag

6 Completeness and Counter-Model Construction

We prove here that when the search procedure introduced in the previous section arrives at a stuck sequent, we can immediately read off from the sequent a Kripke counter-model for it. This result is then instrumental in achieving the equivalence between derivability in **L** and validity.

Theorem 2 (Counter-models at stuck sequents). *Let $\mathcal{B}$ be a failed branch of a proof attempt with top sequent $\Gamma \vdash_G \Delta$.*

1. *The structure $K = (W, \leq, I)$ where W is the set of labels in the sequent, $\leq$ is the reflexive-transitive closure of G and $I(x) = \{p \mid x : p^\circ \in \Gamma\}$, is a Kripke structure.*
2. *Let G_{ext} stand for G extended with all pairs removed in loop steps of $\mathcal{B}$[2]. Let v be the valuation on W such that $v(y) = x$ if (x, y) or (y, x) is in G_{ext} and*

[2] Here is assumed that when an eigenlabel is created at a branch of a proof attempt it is distinct of any other label occurring below in the branch.

$y \notin G$; and $v(y) = y$ otherwise. For any sequent $\Gamma' \vdash_{G'} \Delta'$ in $\mathcal{B}$, (i) for all $xG'y$, $v(x) \leq v(y)$, and moreover (ii) for any formula A,

(a) if $x : A$ or $x : A^\circ$ or $x : A^\bullet$ belongs to Γ', $v(x) \models A$;

(b) if $x : A^* \in \Gamma'$, for all $w \in W$ s.t. $v(x) \leq^{-}_{G'} w$, $w \models A$;

(c) if $x : A$ or $x : A^\circ$ or $x : A^\bullet$ belongs to Δ', $v(x) \not\models A$;

(d) if $x : A^* \in \Delta'$, for all $w \in W$ s.t. $w \leq^{-}_{G'} v(x)$, $w \not\models A$;

hence, in particular, (K, v) is a cm of $\Gamma' \vdash_{G'} \Delta'$.

Proof: 1. If $x = x_1Gx_2G...Gx_n = y$ and $x : p^\circ \in \Gamma$, follows by induction on n that $y : p^\circ \in \Gamma$ (using Lemma 3 in the step case).

2. (i) Trivial. (ii) By induction on the formula A and sub-induction on the number of sequents above $\Gamma' \vdash_{G'} \Delta'$ in $\mathcal{B}$. If $\Gamma' \vdash_{G'} \Delta'$ is $\Gamma \vdash_G \Delta$ itself, it can only have circled atoms or else starred or bulleted formulas. The conditions on circled atoms hold by construction of (K, v) and the conditions on starred formulas hold trivially, since $\leq^{-}_{G} = \leq \setminus v(G)^* = \emptyset$. If $x : (C \prec D)^\bullet \in \Gamma'$ (for $x : (C \supset D)^\bullet \in \Delta'$ the argument is analogous), it can be proved that there are labels y and z, such that $zG_{\mathsf{ext}}y(G_{\mathsf{ext}})^*x$ and $\mathcal{B}$ includes a step

$$\frac{\Gamma'', y : (C \prec D)^\bullet, z : C \vdash_{G'' \cup \{(z,y)\}} z : D, \Delta''^{z/y}, \Delta''}{\Gamma'', y : C \prec D \vdash_{G''} \Delta''} \prec L$$

By the outer IH, $v(z) \models C$ and $v(z) \not\models D$. Thus, since $zG_{\mathsf{ext}}y(G_{\mathsf{ext}})^*x$ implies $v(z) \leq v(y) \leq v(x)$, $v(x) \models C \prec D$.

If $\Gamma' \vdash_{G'} \Delta'$ is the conclusion of an atom or logical inference, the desired conditions follow by the inner IH applied to the premise in $\mathcal{B}$. Let us consider the case of the *loopUp* rule (*loopDn* is analogous):

$$\frac{y \notin G_0 \quad \Gamma' \setminus \Gamma'(y) \vdash_{G_0} \Delta'[x_0/y]}{\Gamma' \vdash_{G_0 \cup \{(x_0,y)\}} \Delta'} \; loopUp$$
$$\text{provided } \Gamma'[y] \subseteq \Gamma'[x_0] \cup \Gamma'^\bullet[x_0], \Gamma'^*[y] \subseteq \Gamma'^*[x_0], \Gamma'^\circ[y] \subseteq \Gamma'^\circ[x_0]$$

Conditions (a) and (b) restricted to $\Gamma' \setminus \Gamma'(y)$ hold by the inner IH. As to $\Gamma'(y)$: for $y : A^\diamond \in \Gamma'$ ($\diamond \in \{*, \circ\}$), the proviso guarantees $x_0 : A^\diamond \in \Gamma'$ and so, by the inner IH and $v(y) = v(x_0)$, $v(y) \models A$; for $y : A \in \Gamma'$, the proviso guarantees either $x_0 : A \in \Gamma'$ or $x_0 : A^\bullet \in \Gamma'$, but both cases follow also from the inner IH[3]. Conditions (c) and (d) follow from the inner IH and the facts $\Delta'(y) \subseteq (\Delta'[x_0/y])(x)$ and $v(y) = v(x_0)$. □

Corollary 1. *1. Let $\Gamma \vdash_G \Delta$ be an* **L***-sequent whose graph is a tree. The following statements are equivalent: i) $\Gamma \vdash_G \Delta$ is derivable in* **L***; ii) $\Gamma \vdash_G \Delta$ is valid; iii) the attempt to prove $\Gamma \vdash_G \Delta$ terminates with success.*

2. For any **L***-sequent whose graph is a tree, the search procedure yields either a proof or a counter-model.*

[3] The second case illustrates why the inductive argument does not go through, if we simply prove that (K, v) is a cm of $\Gamma' \vdash_{G'} \Delta'$.

Proof: 1. Follows from Thm. 2 with the help of Thm. 1 and Prop. 1.

2. Apply the search procedure to the given sequent. Thm. 1 guarantees that it terminates. If this happens with success, then by Prop. 4 the sequent is provable in **L**. Otherwise, the proof attempt has at least one failed branch and thus Thm. 2 guarantees the wanted cm. □

7 Conclusion

Although bi-intuitionistic logic may seem to be a modest extension of intuitionistic logic, it has proved to be rather intricate from the structural proof theory point of view. While naive sequent calculus formalizations are incomplete without a cut rule, more considerate attempts at the design of sequent calculi for backward proof search seem all to lead to relatively sophisticated designs.

We believe that our labelled sequent calculus represents a meaningful compromise between declarativeness and algorithmicity by encoding a reasonably straightforward Kripke semantics based search strategy very much in the spirit of analytic tableaux. Some novelties include integration of all useful monotonicity consequences into the logical rules, including a specific annotation to deal with consequences that must be delayed (flow of information into worlds not yet created), and a termination argument utilizing the fact that information cannot flow around too many turns. The failure-collecting sequent calculus by Postniece and Goré [1,7] and the new calculus of nested sequents by Goré et al. [8] are systems with the same aim and we find the nested sequent calculus especially neat proof-theoretically, although it may require fine-tuning to be practical in theorem proving/counter-model building.

As future work, we would like to see whether bi-intuitionistic logic admits a loop-free backward-search proof system à la Dyckhoff [4], possibly modifiable into a refutation system [13]. We would like to see if it is possible to devise a system with controlled (“analytic”) cuts by a careful analysis of the failure of cut elimination for the Dragalin-style sequent calculus. A yet further line would be to devise a sequent calculus for forward search (a calculus of Mints-style resolution) [10].

On a different note, we would also very much like to come to an understanding of the computational significance of bi-intuitionistic logic, i.e., whether it admits useful a Curry-Howard interpretation justified by a well-motivated, non-degenerate categorical semantics. The first step in this direction was made already by Filinski [5] and further considerations appear in the work of Curien and Herbelin [3]. Crolard's project [2] clearly had the same ultimate aim. We expect that the nested sequences technique of Goré et al. [8] can point to the right structures.

Acknowledgements. We are very grateful to Sara Negri and Linda Postniece for discussions and to our referees for very useful comments. Both of us benefitted from the support of the FP6 IST coordination action TYPES. In addition, L. Pinto was supported by the Portuguese FCT through Centro de Matemática da Universidade do Minho and project RESCUE no. PTDC/EIA/65862/2006 and T. Uustalu was supported by the Estonian Science Foundation through grants no. 5567 and 6940.

References

1. Buisman (Postniece), L., Goré, R.: A cut-free sequent calculus for bi-intuitionistic logic. In: Olivetti, N. (ed.) TABLEAUX 2007. LNCS, vol. 4548, pp. 90–106. Springer, Heidelberg (2007)
2. Crolard, T.: Subtractive logic. Theor. Comput. Sci. 254(1–2), 151–185 (2001)
3. Curien, P.-L., Herbelin, U.: The duality of computation. In: Proc. of 5th Int. Conf. on Functional Programming, ICFP 2000, Montreal, September 2000, pp. 233–243. ACM Press, New York (2000)
4. Dyckhoff, R.: Contraction-free sequent calculi for intuitionistic logic. J. of Symb. Logic 57(3), 795–807 (1992)
5. Filinski, A.: Declarative continuations: An investigation of duality in programming language semantics. In: Dybjer, P., Pitts, A.M., Pitt, D.H., Poigné, A., Rydeheard, D.E. (eds.) Category Theory and Computer Science. LNCS, vol. 389, pp. 224–249. Springer, Heidelberg (1989)
6. Goré, R.: Dual intuitionistic logic revisited. In: Dyckhoff, R. (ed.) TABLEAUX 2000. LNCS, vol. 1847, pp. 252–267. Springer, Heidelberg (2000)
7. Goré, R., Postniece, L.: Combining derivations and refutations for cut-free completeness in bi-intuitionistic logic. J. of Logic and Comput. (to appear)
8. Goré, R., Postniece, L., Tiu, A.: Cut-elimination and proof-search for bi-intuitionistic logic using nested sequents. In: Areces, C., Goldblatt, R. (eds.) Advances in Modal Logic 7, pp. 43–66. College Publications, London (2008)
9. Łukowski, P.: Modal interpretation of Heyting-Brouwer logic. Bull. of Sect. of Logic 25, 80–83 (1996)
10. Mints, G.: Gentzen-type systems and resolution rules, part 1: Propositional logic. In: Martin-Löf, P., Mints, G. (eds.) COLOG 1988. LNCS, vol. 417, pp. 198–231. Springer, Heidelberg (1990)
11. Monteiro, C.: Caracterizações semânticas e dedutivas da lógica bi-intuitionista. Master's thesis. Universidade de Trás-os-Montes e Alto-Douro (2006)
12. Negri, S.: Proof analysis in modal logic. J. of Philos. Logic 34(5–6), 507–544 (2005)
13. Pinto, L., Dyckhoff, R.: Loop-free construction of counter-models for intuitionistic propositional logic. In: Behara, M., Fritsch, R., Lintz, R.G. (eds.) Proc. of 2nd Gauss Symp., Munich, August 1993, pp. 225–232. Conf. A. Walter de Gruyter, Berlin (1995)
14. Rauszer, C.: Semi-boolean algebras and their applications to intuitionistic logic with dual operators. Fund. Math. 83, 219–249 (1974)
15. Rauszer, C.: A formalization of the propositional calculus of H-B logic. Studia Logica 33(1), 23–34 (1974)
16. Rauszer, C.: Applications of Kripke models to Heyting-Brouwer logic. Studia Logica 36(1–2), 61–71 (1977)
17. Restall, G.: Extending intuitionistic logic with subtraction (unpublished note, 1997), http://consequently.org/writing/extendingj/

Automated Synthesis of Tableau Calculi

Renate A. Schmidt and Dmitry Tishkovsky

School of Computer Science, The University of Manchester

Abstract. This paper presents a method for synthesising sound and complete tableau calculi. Given a specification of the formal semantics of a logic, the method generates a set of tableau inference rules which can then be used to reason within the logic. The method guarantees that the generated rules form a calculus which is sound and constructively complete. If the logic can be shown to admit finite filtration with respect to a well-defined first-order semantics then adding a general blocking mechanism produces a terminating tableau calculus. The process of generating tableau rules can be completely automated and produces, together with the blocking mechanism, an automated procedure for generating tableau decision procedures. For illustration we show the workability of the approach for propositional intuitionistic logic.

1 Introduction

We are interested in the problem of automatically generating a tableau calculus for a logic. We assume that the logic is defined by a high-level specification of the formal semantics. Our aim is to turn this into a set of inference rules that provide a sound and complete tableau calculus for the logic. For a decidable logic we want to generate a terminating calculus. In previous work we have described a framework for turning sound and complete tableau calculi into decision procedures [6]. The prerequisites for this to work are that the logic admits the effective finite model property shown by a filtration argument, and that (i) the tableau calculus is sound and constructively complete, and (ii) a weak form of subexpression property holds for tableau derivations. Constructive completeness is a slightly stronger notion than completeness and means that for every open branch in a tableau there is a model which reflects all the expressions (formulae) occurring on the branch. The subexpression property says that every expression in a derivation is a subexpression of the input expression with respect to a finite subexpression closure operator.

In order to be able to exploit the 'termination through blocking' results in [6], in this paper, our goal is to produce tableau calculi that satisfy the prerequisites (i) and (ii). It turns out that provided that the semantics of the logic is well-defined in a certain sense, the subexpression property can be imposed on the generated calculus. Crucial is the separation of the syntax of the logic from the 'extras' in the meta-language needed for the semantic specification of the logic. The process can be completely automated and gives, together with the unrestricted blocking mechanism and the results in [5,6], an automated procedure for generating tableau decision procedures for logics, whenever they have

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 310–324, 2009.

the effective finite model property with respect to a well-defined first-order semantics.

That the generated calculi are constructively complete has the added advantage that models can be effectively generated from open, finished branches in tableau derivations. This means that the synthesised tableau calculi can be used for model building purposes.

The method works as follows. The user defines the formal semantics of the given logic in a many-sorted first-order language so that certain well-definedness conditions hold. The method automatically reduces the semantic specification of the logic to Skolemised implicational forms which are then rewritten as tableau inference rules. Combined with some default closure and equality rules, this provides a sound and constructively complete calculus for the logic. Under certain conditions it is then possible to refine the rules. If the logic can be shown to admit finite filtration, then the generated calculus can be automatically turned into a terminating calculus by adding the unrestricted blocking mechanism from [5].

The method is intended to be as general as possible, and cover as many logics as possible. Our main applications are non-classical logics and description logics. As a case study we consider the application of the method to propositional intuitionistic logic (e.g. [3]). Intuitionistic logic provides a nearly perfect example because the semantics of the logical connectives is not Boolean and the semantics is restricted by a background theory. In addition, the logic is simple.

The paper is structured as follows. Section 2 defines the apparatus for specifying the semantics of the logic of interest. Section 3 is about synthesising tableau rules. In Section 4 we prove that the generated rules form a sound and constructively complete calculus for the logic. Section 5 discusses ways of refining the rules in order to reduce branching and redundancy in the syntax of the calculus. In Section 6 the approach is applied to intuitionistic logic. We conclude with a discussion of the method.

This paper contains no proofs, but these are given in the long version [7]. The long version also contains more examples.

2 Specifying the Semantics of the Logic

For the sake of generality we assume the logic for which we want to develop a tableau calculus is a many-sorted logic.

Let $\mathsf{Sorts} \overset{\mathsf{def}}{=} \{0, 1, \ldots, N\}$ be an index set of sorts and Conn a countable set of the logical connectives of the logic. Every connective σ in Conn is associated with a tuple $(i_1, i_2, \ldots, i_{m+1}) \in \mathsf{Sorts}^{(m+1)}$, where $m \geq 0$. The last argument i_{m+1} is the sort of the expression obtained by applying σ to expressions of sorts $i_1, i_2, \ldots, i_m$, respectively. We say that σ is an m-ary connective of sort $(i_1, i_2, \ldots, i_{m+1})$.

By $\mathcal{L}$ we denote an *abstract sorted language* defined over an alphabet given by a set of sorts Sorts, a set of connectives Conn, a countable set of variable symbols $\{p^i_j \mid i \in \mathsf{Sorts}, j \in \omega\}$, and a countable set of constant symbols $\{q^i_j \mid i \in \mathsf{Sorts}, j \in \omega\}$. $\mathcal{L}$ is defined as a set of *expressions* over the alphabet closed under the connectives in *Conn*. More formally, let $\mathcal{L} \overset{\text{def}}{=} \bigcup_{i \in \mathit{Sorts}} \mathcal{L}^i$, where each $\mathcal{L}^i$ denotes a *set of expressions of sort* i defined as the smallest set of expressions satisfying the following conditions:

- All variables p^i_j and all constants q^i_j in the alphabet belong to $\mathcal{L}^i$.
- For every connective $\sigma \in \mathit{Conn}$ of sort $(i_1, i_2 \ldots, i_{m+1})$, $\sigma(E_1, \ldots, E_m)$ belongs to $\mathcal{L}^{i_{m+1}}$, whenever $E_1, \ldots, E_m$ belong to $\mathcal{L}^{i_1}, \ldots, \mathcal{L}^{i_m}$, respectively.

Symbols, expressions and connectives in $\mathcal{L}$ are also referred to as $\mathcal{L}$-symbols, $\mathcal{L}$-expressions and $\mathcal{L}$-connectives. Variables and constants in $\mathcal{L}$ are called *atomic* $\mathcal{L}$-expressions. We usually refer to expressions in $\mathcal{L}^0$ as *individuals*, expressions in $\mathcal{L}^1$ as *concepts*, and expressions in $\mathcal{L}^2$ as *roles*.

For an $\mathcal{L}$-expression E, the notation $E(p_1, \ldots, p_m)$ indicates that $p_1, \ldots, p_m$ are variables in the expression E. $E(E_1, \ldots, E_m)$ denotes the expression obtained by uniformly substituting E_i into p_i, for all $i = 1, \ldots, m$. Similarly, if X is a set of $\mathcal{L}$-expressions depending on variables $p_1, \ldots, p_m$, we indicate this as $X(p_1, \ldots, p_m)$ and denote by $X(E_1, \ldots, E_m)$ the set of expressions which are instances of expressions from X under uniform substitution of expressions $E_1, \ldots, E_m$ into $p_1, \ldots, p_m$, respectively.

Let $\prec$ be any transitive ordering on $\mathcal{L}$-expressions, E an $\mathcal{L}$-expression, and X a set of $\mathcal{L}$-expressions. We define $\mathsf{sub}_{\prec}(E) \overset{\text{def}}{=} \{E' \mid E' \prec E\}$ and $\mathsf{sub}_{\prec}(X) \overset{\text{def}}{=} \bigcup_{E \in X} \mathsf{sub}_{\prec}(E)$. We write $\mathsf{sub}_{\prec}(E_1, \ldots, E_m)$ rather than $\mathsf{sub}_{\prec}(\{E_1, \ldots, E_m\})$.

The language in which the semantics of the given logic is specified, is a sorted first-order language with equality, denoted by $\mathit{FO}(\mathcal{L})$. Formally, $\mathit{FO}(\mathcal{L})$ is an extension of the language $\mathcal{L}$ with: one additional sort, additional symbols, the usual connectives and quantifiers of first-order logic, and the equality predicate. The sorts of $\mathit{FO}(\mathcal{L})$ are $\mathit{Sorts} \cup \{N+1\} = \{0, \ldots, N, N+1\}$. We call the additional sort $N+1$ the *designated sort*, and symbols that operate on this sort, *designated symbols*. The additional symbols comprise of a countable set of variable symbols $\{x, y, z, x_0, y_0, z_0, \ldots\}$ of the designated sort, a countable set of constants $\{a, b, c, a_0, b_0, c_0, \ldots\}$ of the designated sort, function symbols $\{f, g, h, f_0, g_0, h_0, \ldots\}$ mapping argument terms to terms of sort $N+1$, and a countable set of constant predicate symbols $\{P, Q, R, P_0, Q_0, R_0, \ldots\}$ of the designated sort (i.e., argument terms are required to be terms of sort $N+1$). In addition, $\mathit{FO}(\mathcal{L})$ contains intersort symbols denoted by $\nu_0, \ldots, \nu_N$, i.e., one for each sort in *Sorts*. The purpose of the intersort symbols is to define the semantics of the connectives of the logic (similar to satisfaction conditions in standard definitions of the semantics of connectives). In particular, ν_0 is a unary function symbol of sort $(0, N+1)$, and each of the remaining ν_i is a predicate symbol of sort $(i, N+1, \ldots, N+1)$, with arity $i+1$. Furthermore, for every sort we assume the presence of a binary predicate symbol functioning as equality predicate for that sort. For reasons of simplicity, we use one symbol, namely $\approx$, for each of the equality predicates.

We fix some more common notation. $\overline{w}$ denotes a sequence of first-order variables: $\overline{w} \overset{\text{def}}{=} w_1, \ldots, w_n$. Similarly, $\forall\overline{w}$ denotes the universal quantifier prefix $\forall\overline{w} \overset{\text{def}}{=} \forall w_1 \cdots \forall w_n$. For any set S of formulae, $\forall S$ denotes the universal closure of S, i.e., the set $\forall S \overset{\mathsf{def}}{=} \{\forall \overline{w}\ \phi(\overline{w}) \mid \phi(\overline{w}) \in S\}$. The symbol $\sim$ denotes complementation, i.e., $\sim\psi$ denotes ψ' if $\psi = \neg\psi'$, and $\neg\psi$, otherwise.

Formulae of $\mathsf{FO}(\mathcal{L})$ in which all occurrences of the $\mathcal{L}$-variables p^i_j (of sorts $i = 0, \dots, N$) are free are called *$\mathcal{L}$-open* formulae. Similarly, any $\mathcal{L}$-open formula is an *$\mathcal{L}$-open sentence* if it does *not* have free occurrences of variables of the designated sort $N+1$.

For any set S of $\mathcal{L}$-open formulae in $\mathsf{FO}(\mathcal{L})$ and a set X of $\mathcal{L}$-expressions, let $S{\restriction}X$ be the set of substitution instances of formulae in S under substitutions into the variables of $\mathcal{L}$ which do not contain expressions outside X. Formally,

$$\begin{aligned} S{\restriction}X \overset{\mathsf{def}}{=} \{\phi(E_1, \dots, E_m) \mid{}& \phi(p_1, \dots, p_m) \in S \text{ and} \\ & \text{all } \mathcal{L}\text{-expressions occurring in } \phi(E_1, \dots, E_m) \text{ belong to } X\}. \end{aligned}$$

The semantics of $\mathcal{L}$ is specified in $\mathsf{FO}(\mathcal{L})$ as follows. Each expression in $\mathcal{L}$ is interpreted as a term in $\mathsf{FO}(\mathcal{L})$. In particular, each variable symbol p^i_j in $\mathcal{L}^i$ is interpreted as a variable of sort i in $\mathsf{FO}(\mathcal{L})$, each constant symbol q^i_j in $\mathcal{L}^i$ is interpreted as a constant of sort i in $\mathsf{FO}(\mathcal{L})$, and every connective σ is interpreted as a function of the same sort as σ.

An *$\mathcal{L}$-structure* is a tuple $\mathcal{I} \overset{\mathsf{def}}{=} (\mathcal{L}^0, \dots, \mathcal{L}^N, \Delta^{\mathcal{I}}, \nu_0^{\mathcal{I}}, \dots, \nu_N^{\mathcal{I}}, a^{\mathcal{I}}, \dots, P^{\mathcal{I}}, \dots)$ where $\Delta^{\mathcal{I}}$ is a non-empty set, $\nu_0(\ell)^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for every individual $\ell \in \mathcal{L}^0$, $\nu_n^{\mathcal{I}} \subseteq \mathcal{L}^n \times (\Delta^{\mathcal{I}})^n$, for $0 < n \leq N$. $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ and $P^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^m$, where m is the arity of P. Observe that an $\mathcal{L}$-structure $\mathcal{I}$ is a first-order model (interpretation) of the language $\mathsf{FO}(\mathcal{L})$. For simplicity we omit the sets $\mathcal{L}^0, \dots, \mathcal{L}^N$ and simply write $\mathcal{I} = (\Delta^{\mathcal{I}}, \nu_0^{\mathcal{I}}, \dots, \nu_N^{\mathcal{I}}, a^{\mathcal{I}}, \dots, P^{\mathcal{I}}, \dots)$.

A *valuation* in $\mathcal{I}$ is a mapping ι from the set of variables and constants of $\mathsf{FO}(\mathcal{L})$ to $\mathcal{L} \cup \Delta^{\mathcal{I}}$ such that $\iota(p^i_j), \iota(q^i_j) \in \mathcal{L}^i$, and $\iota(x_j), \iota(a_j) \in \Delta^{\mathcal{I}}$. We say that a valuation ι in an $\mathcal{L}$-structure is *canonical* if every variable and constant of any sort $i = 0, \dots, N$ is interpreted by itself, that is, $\iota(p^i_j) = p^i_j$ and $\iota(q^i_j) = q^i_j$ for every variable p^i_j and constant q^i_j of the language $\mathcal{L}$. This means that a canonical valuation of every term of any sort $i = 0, \dots, N$ is the term itself. It is not difficult to see that any $\mathcal{L}$-open formula ϕ is satisfiable in an $\mathcal{L}$-structure iff it is satisfiable in an $\mathcal{L}$-structure under a canonical valuation. We write $S \models_c S'$ for sets of formulae S and S', if, for every $\mathcal{L}$-structure $\mathcal{I}$ and a canonical valuation ι in $\mathcal{I}$, $\mathcal{I}, \iota \models S$ implies $\mathcal{I}, \iota \models S'$. Similarly, we write $\mathcal{I} \models_c S$ iff there is a canonical valuation ι such that $\mathcal{I}, \iota \models S$.

We say that a concept C is *satisfiable* in $\mathcal{I}$ if there is an $a \in \Delta^{\mathcal{I}}$ such that $(C, a) \in \nu_1^{\mathcal{I}}$, or equivalently $\mathcal{I} \models_c \exists x\ \nu_1(C, x)$. A concept C is *valid* in $\mathcal{I}$ if $\mathcal{I} \models_c \forall x\ \nu_1(C, x)$.

Let S be a set of $\mathcal{L}$-open sentences in $\mathsf{FO}(\mathcal{L})$. A formula ϕ^σ in the language of S *defines the connective* σ with respect to S if it does not contain σ and the following holds:

$$\forall S \models \forall p_1 \dots \forall p_m\ \forall \overline{x}\ (\nu_n(\sigma(p_1, \dots, p_m), \overline{x}) \equiv \phi^\sigma(p_1, \dots, p_m, \overline{x})). \qquad (*)$$

Here $p_1, \dots, p_m$ are variables of appropriate sorts which match the signature of σ, and n is the result sort of σ (for $\overline{x} = (x_1, \dots, x_n)$). We also say *$S$ defines σ*.

$$\forall x\ (x \approx x) \qquad \forall x \forall y\ (x \approx y \rightarrow y \approx x) \qquad \forall x \forall y \forall z\ (x \approx y \wedge y \approx z \rightarrow x \approx z)$$
$$\forall x_1 \cdots \forall x_n \forall y_i\ \ (P(x_1, \ldots, x_n) \wedge x_i \approx y_i \rightarrow P(x_1, \ldots x_{i-1}, y_i, x_{i+1}, x_n))$$
$$\forall p\, \forall x_1 \cdots \forall x_n \forall y_i\ \ (\nu_n(p, x_1, \ldots, x_n) \wedge x_i \approx y_i \rightarrow \nu_n(p, x_1, \ldots x_{i-1}, y_i, x_{i+1}, x_n))$$
$$\forall p_1 \cdots \forall p_m \forall x_1 \cdots \forall x_n \forall y_i\ (x_i \approx y_i \rightarrow$$
$$f(p_1, \ldots, p_m, x_1, \ldots, x_n) \approx f(p_1, \ldots, p_m, x_1, \ldots x_{i-1}, y_i, x_{i+1}, \ldots, x_n))$$

Fig. 1. Equality axioms in $\mathit{FO}(\mathcal{L})$

The $\mathcal{L}$-open sentence $\forall \overline{x}\ (\nu_n(\sigma(p_1, \ldots, p_m), \overline{x}) \equiv \phi^\sigma(p_1, \ldots, p_m, \overline{x})$ is said to be a *σ-definition (in S)*.

By definition, a *(first-order) semantic specification* of $\mathcal{L}$ is a set of $\mathcal{L}$-open sentences in $\mathit{FO}(\mathcal{L})$ that defines the connectives of $\mathcal{L}$. Given a semantic specification S, we use the notation S^0 for the set of $\mathcal{L}$-open sentences defining the connectives of $\mathcal{L}$.

For the sake of generality, we always include the usual equality axioms, listed in Figure 1, in a semantic specification. This ensures that $\approx$ is a congruence on every sort in any first-order interpretation of $\mathit{FO}(\mathcal{L})$.

We consider a semantic specification S to be *normalised* if it consists of three disjoint parts. More specifically, $S = S^+ \cup S^- \cup S^b$, where S^+, S^-, and S^b are disjoint sets of sentences satisfying the following:

(n1) S^+ is a set of $\mathcal{L}$-open sentences ξ^E_+ of the form:

$$\xi^E_+ \ \stackrel{\mathsf{def}}{=}\ \forall \overline{x}\ (\nu_n(E(p_1, \ldots, p_m), \overline{x}) \rightarrow \phi^E_+(p_1, \ldots, p_m, \overline{x})),$$

(n2) S^- is a set of $\mathcal{L}$-open sentences ξ^E_- of the form:

$$\xi^E_- \ \stackrel{\mathsf{def}}{=}\ \forall \overline{x}\ (\phi^E_-(p_1, \ldots, p_m, \overline{x}) \rightarrow \nu_n(E(p_1, \ldots, p_m), \overline{x})),$$

(n3) None of the $\mathcal{L}$-open sentences in S^b contain non-atomic $\mathcal{L}$-expressions.

In this definition, we suppose that multiple sentences of the form (n1) (resp. (n2)) for the same expression E in S^+ and S^- are all equivalently reduced to a single sentence ξ^E_+ (resp. ξ^E_-). The intuition is that S^+ and S^- define the semantics of the connectives. S^+ defines it for positive occurrences of expressions E (with free variables $p_1, \ldots, p_m$), while S^- defines it for negative occurrences of expressions E. We refer to S^b as the *background theory* of the semantics S. Note that S^b includes the equality axioms.

It can be seen that the set $S^0 \cup S^b$ is a semantic specification which can be turned into normalised form by decomposing each connective definition in S^0 into two implications. In fact, S^0 and $S^+ \cup S^-$ play the same role in axiomatising $\mathcal{L}$-connectives in $\mathit{FO}(\mathcal{L})$ modulo the background theory S^b.

For every $\mathcal{L}$-expression E, let

$$\varPhi^E_+ \ \stackrel{\mathsf{def}}{=}\ \{\phi^F_+(E_1, \ldots, E_m, \overline{x}) \mid E = F(E_1, \ldots, E_m) \text{ for some } \xi^{F(p_1, \ldots, p_m)}_+ \text{ from } S\},$$
$$\varPhi^E_- \ \stackrel{\mathsf{def}}{=}\ \{\phi^F_-(E_1, \ldots, E_m, \overline{x}) \mid E = F(E_1, \ldots, E_m) \text{ for some } \xi^{F(p_1, \ldots, p_m)}_- \text{ from } S\}.$$

Thus, Φ_+^E (resp. Φ_-^E) is the set of instantiations of succedents (resp. antecedents) of positive (resp. negative) specifications in S, where the antecedents (resp. succedents) are unifiable with the given expression E.

The expression specifications in any normalised semantics S induce an ordering $\prec$ on expressions as follows. Let $\prec$ be the smallest transitive ordering satisfying: $E' \prec E$ whenever there is a sentence $\xi_+^{F(p_1,\dots,p_m)}$ or a sentence $\xi_-^{F(p_1,\dots,p_m)}$ such that $E = F(E_1,\dots,E_m)$, for some $\mathcal{L}$-expressions $E_1,\dots,E_m$, and E' occurs in $\phi_+^F(E_1,\dots,E_m,\overline{x})$ or $\phi_-^F(E_1,\dots,E_m,\overline{x})$, respectively. Because we can assume that S^0 is also a normalised semantic specification, it similarly induces an ordering $\prec_0$ which is assumed to be well-founded.

Usually the semantics is defined by induction in terms of definitions of the semantics of the connectives and the primitives in the logic which is lifted to arbitrary $\mathcal{L}$-expressions. This is equivalent to assuming a well-founded ordering on expressions of $\mathcal{L}$. For any reasonable definition such a well-founded ordering exists. Thus, although it is not difficult to imagine formulae ϕ^σ such that $\prec_0$ is not well-founded, we assume that the ϕ^σ are chosen in such a way that it is possible to lift the semantics of $\mathcal{L}$-primitives to all $\mathcal{L}$-expressions, i.e., $\prec_0$ is well-founded.

Recall that S^0 denotes the set of $\mathcal{L}$-open sentences that define the $\mathcal{L}$-connectives. A semantic specification S is *well-defined* iff

(wd1) $\forall S^0, \forall S^b \models \forall S$,

(wd2) the ordering $\prec$ is well-founded, and

(wd3) $\forall S^0, S^b{\restriction}\mathsf{sub}_\prec(\sigma(\overline{p})) \models_c \forall\overline{x}\Big(\Big(\bigwedge \Phi_+^{\sigma(\overline{p})} \to \phi^\sigma(\overline{p},\overline{x})\Big) \wedge$
$$\Big(\phi^\sigma(\overline{p},\overline{x}) \to \bigvee \Phi_-^{\sigma(\overline{p})}\Big)\Big).$$

According to this definition, a well-defined semantics S is equivalent to $S^0 \cup S^b$ modulo the background theory S^b. This is ensured by condition (wd1) and the assumption that S defines all $\mathcal{L}$-connectives in S^0. Through condition (wd2), S imposes its own inductive structure on $\mathcal{L}$-expressions. Condition (wd3) specifies a correlation between S and S^0 on instances of $\mathcal{L}$-expressions. It can be seen that $S^0 \cup S^b$ is a well-defined semantic specification itself.

A *(propositional) logic* L over the language $\mathcal{L}$ is a subset of concepts in $\mathcal{L}$ which is closed under arbitrary substitutions of variables with expressions of the same sorts. A logic L is *first-order definable* iff there is a semantic specification S_L such that L coincides with the set of all concepts that are valid in all $\mathcal{L}$-structures satisfying $\forall S_L$, i.e., $L = \{C \in \mathcal{L}^1 \mid \forall S_L \models_c \forall x\, \nu_1(C,x)\}$.

For a fixed semantic specification S_L of logic L, if $\mathcal{I}$ is an $\mathcal{L}$-structure satisfying S_L then by definition $\mathcal{I}$ is a *model of* L or simply a L*-model* (with respect to S_L).

The following are examples of first-order definable logics, which all have a normalised semantic specification according to the above definitions: most description logics, including $\mathcal{ALCO}$, $\mathcal{ALBO}$ [5], $\mathcal{SHOIQ}$ [1], most propositional modal logics, including K, $\mathsf{S4}$, $\mathsf{KD45}$, propositional intuitionistic logic [3], and the logic of metric and topology [2].

3 Synthesising a Tableau Calculus

A *tableau calculus* is a set of tableau inference rules. A *tableau inference rule* is a rule of the form $X/X_1 \mid \cdots \mid X_m$, where both the numerator X and all denominators X_i ($m \geq 0$) are finite sets of negated or unnegated atomic formulae in the language $\mathsf{FO}(\mathcal{L})$. The formulae in the numerator are called *premises*, while the formulae in the denominators are called *conclusions.* The numerator and all the denominators are non-empty, but m may be zero, in which case the rule is a *closure rule* and is usually written $X/\bot$. If $m > 1$, the rule is a branching rule.

Inference steps are performed as usual. A rule is applied to a set of (ground) literals in a branch of a tableau derivation, if the literals are instances of the premises of the rule. Then, in the case of a non-branching rule, the corresponding (ground) instances of the conclusions of the rule are added to the branch. In the case of a branching rule the branch is split into several branches and the corresponding (ground) instances of the conclusions are added to each branch.

Let T denote a tableau calculus and C a concept. We take an arbitrary constant a of the designated sort which does not occur in the rules of T. We denote by $T(C)$ a finished tableau derivation built by starting with the formula $\nu_1(C, a)$ as input and applying the rules of T. That is, all branches in the tableau derivation are fully expanded and all applicable rules of T have been applied in $T(C)$. As usual we assume that all the rules of the calculus are applied *non-deterministically in a tableau derivation.* A branch of a tableau derivation is *closed* if a closure rule has been applied, otherwise the branch is called *open.* The tableau derivation $T(C)$ is *closed* if all its branches are closed and $T(C)$ is *open* otherwise. The calculus T is *sound* (for L) iff for any concept C, each $T(C)$ is open whenever C is satisfiable in an L-model. T is *complete* iff for any unsatisfiable concept C there is a $T(C)$ which is closed. T is said to be *terminating* if every finished open tableau derivation in T has a finite open branch.

Let L be a first-order definable propositional logic over $\mathcal{L}$ and S_L a well-defined semantic specification of L. We now describe how tableau rules can be synthesised from S_L. If S_L is not already normalised we first normalise it. Thus assume $S_L = S_L^+ \cup S_L^- \cup S_L^b$. Now take a positive specification ξ_+^E in S_L^+. Eliminate quantifiers using Skolemisation and equivalently rewrite ξ_+^E into the following implicational form

$$\nu_n(E(p_1, \ldots, p_m), x_1, \ldots, x_n) \to \bigvee_{j=1}^{J} \bigwedge_{k=1}^{K_j} \psi_{jk},$$

where each ψ_{jk} denotes a literal. This is always possible. The implication is now turned into the rule:

$$\rho_+(\xi_+^E) \overset{\mathsf{def}}{=} \frac{\nu_n(E(p_1, \ldots, p_m), x_1, \ldots, x_n), \quad y_1 \approx y_1, \quad \ldots, \quad y_s \approx y_s}{\psi_{11}, \quad \ldots, \quad \psi_{1K_1} \mid \cdots \mid \psi_{J1}, \quad \ldots, \quad \psi_{JK_J}},$$

where $y_1, \ldots, y_s$ denote the free variables in ψ_{jk} which are not among the variables $x_1, \ldots, x_n$. Essentially, the antecedent of the implication has become the main premise in the nominator and the succedent was appropriately turned into the denominators of the rule. The purpose of the equations $y_i \approx y_i$ is domain predication. We say the rule *corresponds* to ξ_+^E. Analogously a tableau rule is be generated for each negative specification ξ_-^E in S_L^-. The contrapositive of ξ_-^E is equivalently rewritten to Skolemised implicational form

$$\neg\nu_n(E(p_1,\ldots,p_m),x_1,\ldots,x_n) \rightarrow \bigvee_{j=1}^{J} \bigwedge_{k=1}^{K_j} \psi_{jk},$$

where each ψ_{jk} denotes a literal, and the corresponding rules have the form

$$\rho_-(\xi_-^E) \overset{\text{def}}{=} \frac{\neg\nu_n(E(p_1,\ldots,p_m),x_1,\ldots,x_n), \quad y_1 \approx y_1, \quad \ldots, \quad y_s \approx y_s}{\psi_{11}, \quad \ldots, \quad \psi_{1K_1} \mid \cdots \mid \psi_{J1}, \quad \ldots, \quad \psi_{JK_J}}.$$

We refer to the rules $\rho_+(\xi_+^E)$ and $\rho_-(\xi_-^E)$ as *decomposition rules.*

For example, the generated decomposition rules for the existential restriction operator in the description logic $\mathcal{ALC}$ are:

$$\frac{\nu_1(\exists r.p, x)}{\nu_2(r,x,f(p,x)), \quad \nu_1(p,f(p,x))}, \qquad \frac{\neg\nu_1(\exists r.p, x), \quad y \approx y}{\neg\nu_2(r,x,y) \mid \neg\nu_1(p,y)}.$$

These are not the familiar rules used in standard description logic tableau systems, but in Section 5 we see how to get those.

The sentences in the background theory of S_L are turned into rules by first equivalently transforming them into Skolemised disjunctive normal form. More specifically, let ξ be an arbitrary sentence in S_L^b. It is first rewritten to

$$\bigvee_{j=1}^{J} \bigwedge_{k=1}^{K_j} \psi_{jk}, \qquad (**)$$

where each ψ_{jk} denotes a literal, and is then turned into the corresponding rule, namely

$$\rho(\xi) \overset{\text{def}}{=} \frac{p_1 \approx p_1, \quad \ldots, \quad p_m \approx p_m, \quad x_1 \approx x_1, \quad \ldots, \quad x_n \approx x_n}{\psi_{11}, \quad \ldots, \quad \psi_{1K_1} \mid \cdots \mid \psi_{J1}, \quad \ldots, \quad \psi_{JK_J}}.$$

The $p_1,\ldots,p_m,x_1,\ldots,x_n$ are the variables that are free in (**). Rules corresponding to sentences in S_L^b are called *theory rules.*

We use T_L to denote the generated calculus. In summary, it consists of these rules.

(t1) The decomposition rules $\rho_+^\sigma(\xi)$ and $\rho_-^\sigma(\eta)$ corresponding to all positive specifications ξ in S_L^+ and all negative specifications η in S_L^-.

(t2) The theory rules $\rho(\zeta)$ corresponding to all sentences ζ in the background theory S_L^b.

(t3) The *closure rules* (for every $n = 1,\ldots,N$, and every constant predicate symbol P in S_L):

$$\frac{\nu_n(p,\overline{x}), \quad \neg\nu_n(p,\overline{x})}{\perp}, \qquad \frac{P(\overline{x}), \quad \neg P(\overline{x})}{\perp}.$$

4 Ensuring Soundness and Constructive Completeness

It is possible to prove that every rule of the generated calculus T_L preserves satisfiability of $\mathsf{FO}(\mathcal{L})$-formulae. That is, if all premises of a rule are true in an L-model $\mathcal{I}$ (under a canonical valuation) then the conclusions of some branch are also true. This is not difficult to see because the definitions of the rules mimic the specified semantics. Hence:

Theorem 1 (Soundness). *T_L is sound for L, i.e., for every concept C satisfiable in an L-model, any finished tableau derivation $T_L(C)$ is open.*

Now, we prove constructive completeness of T_L. Let $\mathcal{B}$ denote an arbitrary branch in a T_L-tableau derivation. We define the following relation $\sim_{\mathcal{B}}$ with respect to $\mathcal{B}$: $t \sim_{\mathcal{B}} t' \overset{\mathsf{def}}{\iff} t \approx t' \in \mathcal{B}$, for any ground terms t and t' of the designated sort $N+1$ in $\mathcal{B}$. Let $\|t\| \overset{\mathsf{def}}{=} \{t' \mid t \sim_{\mathcal{B}} t'\}$ be the equivalence class of an element t. The presence of the rules generated from the equality axioms ensure that $\sim_{\mathcal{B}}$ is a congruence relation on all designated ground terms in $\mathcal{B}$.

We say a model $\mathcal{I}$, under a (canonical) valuation ι, *reflects* an expression E occurring in a branch $\mathcal{B}$ iff for every ground terms $t_1, \ldots, t_n$ we have that

- $(E, \iota(t_1), \ldots, \iota(t_n)) \in \nu_n^{\mathcal{I}}$ whenever $\nu_n(E, t_1, \ldots, t_n) \in \mathcal{B}$, and
- $(E, \iota(t_1), \ldots, \iota(t_n)) \notin \nu_n^{\mathcal{I}}$ whenever $\neg\nu_n(E, t_1, \ldots, t_n) \in \mathcal{B}$.

Similarly, $\mathcal{I}$ *reflects* predicate constant P from $\mathcal{B}$ under a (canonical) valuation ι in $\mathcal{I}$ iff for every ground terms $t_1, \ldots, t_n$ we have that

- $(\iota(t_1), \ldots, \iota(t_n)) \in P^{\mathcal{I}}$ whenever $P(t_1, \ldots, t_n) \in \mathcal{B}$, and
- $(\iota(t_1), \ldots, \iota(t_n)) \notin P^{\mathcal{I}}$ whenever $\neg P(t_1, \ldots, t_n) \in \mathcal{B}$.

A model $\mathcal{I}$ *reflects* branch $\mathcal{B}$ under a valuation ι if $\mathcal{I}$ reflects all predicate constants and expressions occurring in $\mathcal{B}$ under ι.

A tableau calculus T_L is said to be *constructively complete* (for L) iff for any given concept C that is satisfiable, if $\mathcal{B}$ is an open branch in the tableau derivation $T_L(C)$ then there is an L-model $\mathcal{I}$ such that:

(m1) The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is the set of the equivalence classes $\|t\|$ for each ground term t occuring in $\mathcal{B}$.

(m2) $\mathcal{I}$ reflects $\mathcal{B}$ under the *canonical projection valuation* π defined by $\pi(t) \overset{\mathsf{def}}{=} \|t\|$, for every ground term t occuring in $\mathcal{B}$.

It is clear that if T_L is constructively complete then T_L is complete for L.

Suppose now that S_L is a semantic specification and $\prec_0$ is a well-founded ordering on $\mathcal{L}$-expressions induced by the set S_L^0 of the definitions of the connectives of the form ($\boxed{*}$) with respect to S_L.

Let $\mathcal{B}$ be an open branch in a finished tableau derivation in T_L. Define interpretations of predicate symbols in $\mathcal{I}(\mathcal{B})$ by induction on $\prec_0$ as follows:

- $P^{\mathcal{I}(\mathcal{B})} \overset{\mathsf{def}}{=} \{(\|t_1\|, \ldots, \|t_n\|) \mid P(t_1, \ldots, t_n) \in \mathcal{B}\}$, for every n-ary constant predicate symbol P in S_L.

- For every $n = 1, \ldots, N$ the interpretation $\nu_n^{\mathcal{I}(\mathcal{B})}$ of the ν_n symbols is defined as the smallest subset of $\mathcal{L}^n \times (\Delta^{\mathcal{I}(\mathcal{B})})^n$ satisfying both $(p, \|t_1\|, \ldots, \|t_n\|) \in \nu_n^{\mathcal{I}(\mathcal{B})} \iff \nu_n(p, t_1, \ldots, t_n) \in \mathcal{B}$ and $(\sigma(E_1, \ldots, E_m), \|t_1\|, \ldots, \|t_n\|) \in \nu_n^{\mathcal{I}(\mathcal{B})} \iff \mathcal{I}(\mathcal{B}) \models_c \phi^\sigma(E_1, \ldots, E_m, \|t_1\|, \ldots, \|t_n\|)$, for every variable or constant p of the sort n, every connective σ, and any expressions $E_1, \ldots, E_m$.

A consequence of the definition of $\mathcal{I}(\mathcal{B})$ is that the definitions of the connectives are valid in $\mathcal{I}(\mathcal{B})$, i.e., we have $\mathcal{I}(\mathcal{B}) \models \forall S_L^0$.

It can be proved that $\mathcal{I}(\mathcal{B})$ reflects the branch $\mathcal{B}$ (under the valuation π) by induction on the well-founded ordering $\prec$. As a consequence we obtain constructive completeness.

Theorem 2 (Constructive completeness). *T_L is constructively complete.*

5 Refining the Synthesised Calculus

In order to get inference rules that have better properties, in this section we introduce two refinements.

The first refinement reduces the number of branches of a rule by constraining the rule with additional premises rather than deriving new conclusions. Suppose $r \overset{\text{def}}{=} X/X_1 \mid \cdots \mid X_m$ is a tableau rule of a sound and constructively complete tableau calculus T_L. For some $i \in \{1, \ldots, m\}$ suppose $X_i = \{\psi_1, \ldots, \psi_k\}$. Without loss of generality we can assume that $i = 1$. Consider the rules r_j with $j = 1, \ldots, k$ defined by

$$r_j \overset{\text{def}}{=} \frac{X \cup \{{\sim}\psi_j\}}{X_2 \mid \cdots \mid X_m}.$$

Note that we can drop any domain predication equalities from the numerator when they are not necessary. Let T'_L be the tableau calculus obtained from T_L by replacing rule r by the rules $r_1, \ldots, r_k$. It is clear that T'_L is sound. In general, T'_L is not constructively complete. However the following theorem is true.

Theorem 3. *Let $\mathcal{B}$ be an open branch in a T'_L-tableau. Assume that for every set Y of $\mathcal{L}$-expressions the following holds.*

If all expressions from Y are reflected in $\mathcal{I}(\mathcal{B})$ then for every $E_1, \ldots, E_l \in Y$,

$$X(E_1, \ldots, E_l, t_1, \ldots, t_n) \subseteq \mathcal{B} \textit{ implies}$$
$$\mathcal{I}(\mathcal{B}) \models X_i(E_1, \ldots, E_l, \|t_1\|, \ldots, \|t_n\|) \textit{ for some } i = 1, \ldots, m. \quad (\dagger)$$

Then, $\mathcal{B}$ is reflected in $\mathcal{I}(\mathcal{B})$.

Corollary 1. *If the condition of Theorem 3 holds for every open branch $\mathcal{B}$ of any T'_L-tableau then T'_L is constructively complete.*

Generalising this refinement to moving more than one conclusion up to the numerator is not difficult. The formulation of Theorem 3 does not change then.

Consider the generated rule for negative occurrences of the existential restriction operator given on p. 317. In most description logics it can be transformed to the more often seen rule:

$$\frac{\neg\nu_1(\exists r.p, x), \quad \nu_2(r, x, y)}{\neg\nu_1(p, y)}.$$

In order to preserve constructive completeness, the following condition is usually proved by induction on $\prec$. This, in turn, inductively implies condition (†).

$$\text{If } \neg\nu_1(\exists E.F, t) \in \mathcal{B} \text{ and } \mathcal{I}(\mathcal{B}) \models \nu_2(E, t, t') \text{ then } \neg\nu_1(F, t') \in \mathcal{B}.$$

Another example of a generated rule and a refinement are (for transitive R):

$$\frac{x \approx x, \quad y \approx y, \quad z \approx z}{\neg R(x,y) \mid \neg R(y,z) \mid R(x,z)}, \qquad\qquad \frac{R(x,y), \quad R(y,z)}{R(x,z)}.$$

Condition (†) holds in this case since, by definition of $\mathcal{I}(\mathcal{B})$, $\mathcal{I}(\mathcal{B})$ reflects all atomic predicate constants in the branch $\mathcal{B}$.

The second refinement we describe exploits the expressivity of the logic. Suppose that the language $\mathcal{L}$ of the logic L is expressive enough to represent its own semantics. That is, assume that for every $n = 0, \ldots, N$ and every n-ary predicate constant P occuring in S_L, there are concepts $C_n^+(p, \ell_1, \ldots, \ell_n)$, $C_n^-(p, \ell_1, \ldots, \ell_n)$, $D_P^+(\ell_1, \ldots, \ell_n)$, and $D_P^-(\ell_1, \ldots, \ell_n)$, depending on variable p of sort n and variables $\ell_1, \ldots, \ell_n$ of sort 0, such that

$$\begin{aligned}
&\forall S_L \models \forall x \left(\nu_1(C_n^+(p, \ell_1, \ldots, \ell_n), x) \rightarrow \nu_n(p, \nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right), \\
&\forall S_L \models \forall x \left(\nu_1(C_n^-(p, \ell_1, \ldots, \ell_n), x) \rightarrow \neg\nu_n(p, \nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right), \\
&\forall S_L \models \forall x \left(\nu_1(D_P^+(\ell_1, \ldots, \ell_n), x) \rightarrow P(\nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right), \\
&\forall S_L \models \forall x \left(\nu_1(D_P^-(\ell_1, \ldots, \ell_n), x) \rightarrow \neg P(\nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right).
\end{aligned}$$

In this case we are able to express all tableau rules for L in the language $\mathcal{L}$ itself. We only need to replace every positive occurrence of $\nu_n(E, x_1, \ldots, x_n)$ in T_L with $C_n^+(E, \ell_1, \ldots, \ell_n)$, every (negative) occurrence of $\neg\nu_n(E, x_1, \ldots, x_n)$ in T_L with $C_n^-(E, \ell_1, \ldots, \ell_n)$, and, similarly, all the predicate constants P need to be replaced with occurrences of D_P^+ or D_P^- depending on the polarity of P. In fact, the sort $N+1$ of $\mathsf{FO}(\mathcal{L})$ can be reflected in the sort 0.

A slight difficulty is caused by Skolem functions in $\mathsf{FO}(\mathcal{L})$ occurring in the tableau calculus, since for them there could be no corresponding function symbols in $\mathcal{L}$. It can be solved by introducing new connectives f_g into the language $\mathcal{L}$ for every (Skolem) function and constant g of $\mathsf{FO}(\mathcal{L})$ so that for every $(p_1, \ldots, p_m, \ell_1, \ldots, \ell_n)$, $f_g(p_1, \ldots, p_m, \ell_1, \ldots, \ell_n)$ is a term of the sort 0 and its semantics is defined by $\nu_0(f_g(\overline{p}, \ell_1, \ldots, \ell_n)) \overset{\mathsf{def}}{=} g(\overline{p}, \nu_0(\ell_1), \ldots, \nu_0(\ell_n))$. An alternative is to introduce unique, new individual names (for every $p_1, \ldots, p_m$, $\ell_1, \ldots, \ell_n$) instead of new connectives.

Connective definitions
$\forall x \; \big(\nu_1(\bot, x) \equiv \bot\big)$
$\forall x \; \big(\nu_1(p_1 \wedge p_2, x) \equiv \nu_1(p_1, x) \wedge \nu_1(p_2, x)\big)$
$\forall x \; \big(\nu_1(p_1 \vee p_2, x) \equiv \nu_1(p_1, x) \vee \nu_1(p_2, x)\big)$
$\forall x \; \big(\nu_1(p_1 \rightarrow p_2, x) \equiv \forall y \; \big(R(x, y) \rightarrow (\nu_1(p_1, y) \rightarrow \nu_1(p_2, y))\big)\big)$

Background theory of a partial ordering R and ν_1
$\forall x \; R(x, x)$
$\forall x \forall y \; (R(x, y) \wedge R(y, x) \rightarrow x \approx y)$
$\forall x \forall y \forall z \; (R(x, y) \wedge R(y, z) \rightarrow R(x, z))$
$\forall x \forall y \; \big(\nu_1(p, x) \wedge R(x, y) \rightarrow \nu_1(p, y)\big)$

Fig. 2. Specification of semantics of intuitionistic logic

For example, in the description logic $\mathcal{ALCO}$ with full support of individuals, we can set (for any atomic role r and individual equality):

$$C_2^+(r, \ell_1, \ell_2) \stackrel{\mathsf{def}}{=} \ell_1 : \exists r.\{\ell_2\}, \qquad D_\approx^+(\ell_1, \ell_2) \stackrel{\mathsf{def}}{=} \ell_1 : \{\ell_2\},$$
$$C_2^-(r, \ell_1, \ell_2) \stackrel{\mathsf{def}}{=} \ell_1 : \neg\exists r.\{\ell_2\}, \qquad D_\approx^-(\ell_1, \ell_2) \stackrel{\mathsf{def}}{=} \ell_1 : \neg\{\ell_2\}.$$

Thus, the language of the tableau calculus can be significantly simplified. For instance, the (refined) rules for the existential restriction operator become:

$$\frac{\ell : \exists r.p}{\ell : \exists r.\{f(r, p, \ell)\}, \quad f(r, p, \ell) : p}, \qquad \frac{\ell : \neg\exists r.p, \quad \ell : \exists r.\{\ell'\}}{\ell' : \neg p}.$$

6 Synthesising Tableaux for Intuitionistic Logic

Intuitionistic logic is an example of a logic where ν_n cannot be expressed in the language of the logic. It also provides an example of a logics for which a background theory is an essential part of the definition of the semantics.

The language of intuitionistic logic is a one-sorted language defined over a countable set of propositional symbols $p_1^1, p_2^1, \ldots$, and the standard connectives are $\rightarrow, \vee, \wedge, \bot$. The semantic specification in $\mathit{FO}(\mathcal{L})$ is given in Figure 2 (cf. [3]). R is the designated predicate symbol representing the partial order in the background theory. For intuitionistic logic the orderings $\prec_0$ and $\prec$ coincide. The ordering $\prec$ on subexpressions induced by the semantic definition of the connectives is the smallest ordering satisfying: $E_1 \prec E_1 \sigma E_2$ and $E_2 \prec E_1 \sigma E_2$, for each $\sigma \in \{\rightarrow, \vee, \wedge\}$ and all intuitionistic formulae E_1 and E_2.

The tableau rules generated by our approach are those listed in Figure 3. Together with the equality rules, they form a calculus, which is sound and constructively complete for propositional intuitionistic logic. This is an immediate consequence of Theorems 1 and 2. Refining the generated rules yields the rules listed in Figure 4. Using Theorem 3 we conclude that these rules together with suitably refined equality rules provide a sound and constructively complete tableau calculus for intuitionistic logic.

Decomposition rules:

$$\frac{\nu_1(\bot, x)}{\bot} \qquad \frac{\nu_1(p_1 \wedge p_2, x)}{\nu_1(p_1, x), \quad \nu_1(p_2, x)} \qquad \frac{\neg\nu_1(p_1 \wedge p_2, x)}{\neg\nu_1(p_1, x) \mid \neg\nu_1(p_2, x)}$$

$$\frac{\neg\nu_1(\bot, x)}{\neg\bot} \qquad \frac{\nu_1(p_1 \vee p_2, x)}{\nu_1(p_1, x) \mid \nu_1(p_2, x)} \qquad \frac{\neg\nu_1(p_1 \vee p_2, x)}{\neg\nu_1(p_1, x), \quad \neg\nu_1(p_2, x)}$$

$$\frac{\nu_1(p_1 \rightarrow p_2, x)}{\neg R(x, y) \mid \neg\nu_1(p_1, y) \mid \nu_1(p_2, y)}$$

$$\frac{\neg\nu_1(p_1 \rightarrow p_2, x)}{R(x, f(p_1, p_2, x)), \quad \nu_1(p_1, f(p_1, p_2, x)), \quad \neg\nu_1(p_2, f(p_1, p_2, x))}$$

Theory rules:

$$\frac{x \approx x}{R(x, x)} \qquad \frac{x \approx x, \quad y \approx y}{\neg R(x, y) \mid \neg R(y, x) \mid x \approx y} \qquad \frac{x \approx x, \quad y \approx y, \quad z \approx z}{\neg R(x, y) \mid \neg R(y, z) \mid R(x, z)}$$

$$\frac{p \approx p, \quad x \approx x, \quad y \approx y}{\neg\nu_1(p, x) \mid \neg R(x, y) \mid \nu_1(p, y)}$$

Closure rules:

$$\frac{\nu_1(p, x), \quad \neg\nu_1(p, x)}{\bot} \qquad \frac{R(x, y), \quad \neg R(x, y)}{\bot}$$

Fig. 3. Generated tableau rules for intuitionistic logic

Decomposition rules:

$$\frac{\nu_1(\bot, x)}{\bot} \qquad \frac{\nu_1(p_1 \wedge p_2, x)}{\nu_1(p_1, x), \quad \nu_1(p_2, x)} \qquad \frac{\neg\nu_1(p_1 \wedge p_2, x)}{\neg\nu_1(p_1, x) \mid \neg\nu_1(p_2, x)}$$

$$\frac{\nu_1(p_1 \vee p_2, x)}{\nu_1(p_1, x) \mid \nu_1(p_2, x)} \qquad \frac{\neg\nu_1(p_1 \vee p_2, x)}{\neg\nu_1(p_1, x), \quad \neg\nu_1(p_2, x)} \qquad \frac{\nu_1(p_1 \rightarrow p_2, x), \quad R(x, y), \quad \nu_1(p_1, y)}{\nu_1(p_2, y)}$$

$$\frac{\neg\nu_1(p_1 \rightarrow p_2, x)}{R(x, f(p_1, p_2, x)), \quad \nu_1(p_1, f(p_1, p_2, x)), \quad \neg\nu_1(p_2, f(p_1, p_2, x))}$$

Theory rules:

$$\frac{x \approx x}{R(x, x)} \qquad \frac{R(x, y), \quad R(y, x)}{x \approx y} \qquad \frac{R(x, y), \quad R(y, z)}{R(x, z)} \qquad \frac{\nu_1(p, x), \quad R(x, y)}{\nu_1(p, y)}$$

Closure rules:

$$\frac{\nu_1(p, x), \quad \neg\nu_1(p, x)}{\bot}$$

Fig. 4. Refined tableau rules for intuitionistic logic

A terminating tableau calculus is obtained if the calculus is enhanced with the blocking mechanism of [5,6]. This follows from the results in [6], the soundness and constructive completeness of the calculus, the subexpression property and the fact that intuitionistic logic admits finite filtrations. The calculus can be turned into a deterministic decision procedure using breadth-first search or depth-first search, as we showed in [6].

7 Discussion and Conclusions

The method introduced in this paper automatically produces a sound and constructively complete tableau calculus from a semantic first-order specification of a many-sorted logic. The method is directly applicable to many non-classical logics and covers many types of ground tableau calculi commonly found in the literature. These include two types of tableau calculi for relations satisfying extra theory conditions which can be accommodated either by structural rules or propagation rules.

The results of the paper can be regarded as a mathematical formalisation and generalisation of tableau development methodologies. Our formalisation is based on, and provides the basis for, the implementation of tableau decision procedures for modal and description logics in the MetTeL system [8]. The formalisation separates the creative part of tableau calculus development, which needs to be done by a human developer, and the automatic part of the development process, which can be left to an automated (currently first-order) prover and an automated tableau synthesiser. The creative part is the specification of the logic so that the conditions of well-foundness of the orderings $\prec_0$ and $\prec$ hold. The automatic part deals with verification of the first-order conditions (wd1) and (wd3), and the generation of tableau rules from the (well-defined) semantics provided by the developer. Then, the developer can transform the generated rules to refined form by applying Theorem 3. Refinements are crucial for the success of the method. With the described refinements the calculi that the method generates are equivalent (modulo inessential variations) to the calculi common for intuitionistic logic, $\mathcal{ALCO}$, most other first-order definable description logics, and the common modal logics such as $\mathsf{S4}$ and $\mathsf{S5}$, for example.

For common modal and description logics conditions (wd1) and (wd3) are simple to check, even trivial in many cases. In fact, a developer usually implicitly formalises the logic's semantics S in such way that $S = S^0 \cup S^b$. This is the case for almost all of known logics. If the specification of the semantics satisfies $S = S^0 \cup S^b$ then conditions (wd1) and (wd3) hold trivially and the orderings $\prec_0$ and $\prec$ coincide. This means the ordering used for the specification of the semantics of the logical connectives (which is usually well-founded), is enough for tableau synthesis.

This paper also presents a general method for proving (constructive) completeness of tableau calculi. In addition, the generated rules can be transformed to an optimal form provided that the special condition (†) has been proven by induction on the ordering $\prec$ for the refined calculus.

With enough expressivity for representing the basics of the semantics within the logic it is possible to simplify the language of the tableau. In this case, the obtained calculus is similar to tableau calculi for description logics with full support of individuals, hybrid modal logic, and labelled tableau calculi. Otherwise, the calculus has the same flavour as the standard tableau calculus for intuitionistic logic, where every node of a tableau is characterised by two complementary sets of true and false formulae (concepts).

As a case study we considered tableau synthesis for propositional intuitionistic logic. We believe the approach is also applicable to most known, first-order definable modal and description logics. Non first-order translatable logics such as propositional dynamic logic are currently beyond the scope of the method.

The tableau calculi generated are Smullyan-type tableau calculi, i.e., ground semantic tableau calculi. We believe that other types of tableau calculi can be generated using the same techniques. Exploiting the known relationships to other deduction methods and the underlying ideas of [4] we expect synthesis of non-tableau approaches is possible as well, but this is future work. In [4] we have shown that it is possible to synthesise tableau calculi for modal logics by translation to first-order logic combined with first-order resolution. In this framework the semantic specification of a logic is transformed into clausal form and then a set of inference rules. Soundness and completeness of the generated calculus follows from the soundness and completeness of the simulating resolution refinement used. This approach has several advantages, but in this paper we have taken a different, more direct approach. Rather than proceeding via simulation by resolution we have shown that tableau rules can be generated directly from the specification of the logic.

Our future goal is to further reduce human involvement in the development of calculi by finding appropriate automatically verifiable conditions for optimal calculi to be generated.

References

1. Horrocks, I., Sattler, U.: A tableau decision procedure for SHOIQ. J. Automat. Reasoning 39(3), 249–276 (2007)
2. Hustadt, U., Tishkovsky, D., Wolter, F., Zakharyaschev, M.: Automated reasoning about metric and topology (System description). In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS(LNAI), vol. 4160, pp. 490–493. Springer, Heidelberg (2006)
3. Kripke, S.A.: Semantical analysis of intuitionistic logic I. In: Formal Systems and Recursive Functions, pp. 92–130. North-Holland, Amsterdam (1965)
4. Schmidt, R.A.: Developing modal tableaux and resolution methods via first-order resolution. In: Advances in Modal Logic, vol. 6, pp. 1–26. College Publ. (2006)
5. Schmidt, R.A., Tishkovsky, D.: Using tableau to decide expressive description logics with role negation. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 438–451. Springer, Heidelberg (2007)
6. Schmidt, R.A., Tishkovsky, D.: A general tableau method for deciding description logics, modal logics and related first-order fragments. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 194–209. Springer, Heidelberg (2008)
7. Schmidt, R.A., Tishkovsky, D.: Automated synthesis of tableau calculi (2009), `http://www.cs.man.ac.uk/~dmitry/papers/astc2009.pdf`
8. Tishkovsky, D.: MetTeL system, `http://www.cs.man.ac.uk/~dmitry/implementations/MetTeL/`

Tableaux for Projection Computation and Knowledge Compilation

Christoph Wernhard

Technische Universität Dresden
christoph.wernhard@tu-dresden.de

Abstract. Projection computation is a generalization of second-order quantifier elimination, which in turn is closely related to the computation of forgetting and of uniform interpolants. On the basis of a unified view on projection computation and knowledge compilation, we develop a framework for applying tableau methods to these tasks. It takes refinements from performance oriented systems into account. Formula simplifications are incorporated at the level of tableau structure modification, and at the level of simplifying encountered subformulas that are not yet fully compiled. In particular, such simplifications can involve projection computation, where this is possible with low cost. We represent tableau construction by means of rewrite rules on formulas, extended with some auxiliary functors, which is particularly convenient for formula transformation tasks. As instantiations of the framework, we discuss approaches to propositional knowledge compilation from the literature, including adaptions of DPLL, and the hyper tableau calculus for first-order clauses.

1 Introduction

There are two established families of methods for second-order quantifier elimination, the SCAN approach, based on resolvent generation, and direct methods, based and Ackermann's lemma [9]. Over the last decade, several methods for second-order quantifier elimination have been proposed that can be classified as belonging to a third approach: tableau construction [12,4,14,16,11]. All of these are restricted to propositional logic (and thus, more specifically perform Boolean quantifier elimination) and most of them are based on DPLL which we regard as a tableau procedure, considering semantic trees as tableaux. They are combined with *knowledge compilation*, the transformation of formulas such that they meet syntactic criteria which permit to execute certain operations with low cost. They are described with different techniques, including pseudocode as common in the literature on modern DPLL methods, and with varying terminology such as *marginalization*, *computation of uniform interpolants*, *forgetting* and *projection computation* for second-order quantifier elimination or closely related tasks. In this paper, we stick with *projection computation*, specified explicitly as a generalization of second-order quantifier elimination.

Our goal in this paper is to develop a framework that realizes abstractions in two respects: First, it provides a unified view on projection computation and

M. Giese and A. Waaler (Eds.): TABLEAUX 2009, LNAI 5607, pp. 325–340, 2009.

certain kinds of knowledge compilation. Second, it abstracts from differences in tableau construction methods that are inessential for the adaption to such tasks. Adaptions of the methods of successful automated tableau systems, such as modern SAT solvers or hyper tableau systems, can then be modeled as instantiations of the framework.

The basic idea is that the whole tableau itself is the objective of computation – in contrast to a single branch representing a model, or a proof of unsatisfiability where the tableau represents the search trace. A computed tableau then represents a transformed input formula that satisfies syntactic criteria which permit certain operations – especially projection computation – to be performed with low cost.

The paper is structured as follows: Preliminaries on projection and the considered compilation target format, linkless formulas, are given in Sect. 2, followed by the definition of a relation that generalizes the relationships between inputs and outputs of projection computation, knowledge compilation, and their combinations. In Sect. 3, a tableau variant is presented that is particularly suited for formula transformation tasks, since tableaux are represented as formulas with some additional structure indicating operators. Tableau construction rules then closely resemble formula rewriting rules. A tableau calculus is introduced and – based on the relation defined in Sect. 2 – shown to satisfy correctness properties with respect to projection computation and knowledge compilation. Refinements of the calculus that seem essential for practical application, such as and-nodes, tableau level simplification and backjumping, are discussed in Sect. 4. Methods for projection computation and knowledge compilation that are based on well known practical successful methods such as DPLL and hyper tableaux are then modeled in Sect. 5 as instances of this tableau framework.

Notation. Unless specially noted, we assume that a formula is in negation normal form, constructed from first-order literals, truth value constants $\top, \bot$, binary connectives $\land, \lor$ and the universal first-order quantifier $\forall$. N-ary connectives are understood as meta-level notation with respect to this syntax. We write the positive (negative) literal with atom A as $+A$ ($-A$) and the complement of literal L as $\widetilde{L}$. As usual, a *sentence* is a formula without free variables. A *universal literal* is a sentence of the form $\forall x_1 \ldots \forall x_n\ L$, where $n \geq 0$ and L is a literal. The symbol LIT denotes the set of all universal literals. We assume a fixed first-order signature Σ and refer to the ground atoms and ground literals constructable from it as *all* ground atoms and ground literals, respectively. The symbol ALL denotes the set of all ground literals, POS all positive ground literals. The *atom base* $\mathcal{A}(F)$ (*literal base* $\mathcal{L}(F)$) of a formula F is the set of all ground atoms (ground literals) which are instance of an of atom (literal) in F.

Semantic Framework. We use the notational variant of Herbrand interpretations described in [19]: An *interpretation* is a pair $\langle I, \beta \rangle$, where I is a *structure*, that is, a set of ground literals that contains for all ground atoms A exactly one of $+A$ or $-A$, and β is a *variable assignment*, that is, a mapping of the set of variables into the set of ground terms. The set of literals I of an interpretation $\langle I, \beta \rangle$ is called *"structure"*, since it represents a structure in the conventional sense used in model theory: The domain is the set of ground terms. Function symbols f with arity $n \geq 0$ are mapped to functions f' such that for all ground terms $t_1, ..., t_n$ it holds that $f'(t_1, ..., t_n) = f(t_1, ..., t_n)$. Predicate symbols p with arity $n \geq 0$ are mapped to $\{\langle t_1, ..., t_n\rangle \mid +p(t_1, ..., t_n) \in I\}$. Moreover, an interpretation $\langle I, \beta\rangle$ represents a conventional second-order interpretation [8] (if predicate variables are considered as distinguished predicate symbols): The structure in the conventional sense corresponds to I, as described above, except that mappings of predicate variables are omitted. The assignment is β, extended such that all predicate variables p are mapped to $\{\langle t_1, ..., t_n\rangle \mid +p(t_1, ..., t_n) \in I\}$.

The satisfaction relation is defined as usual by a clause for each logic operator, for example: $\langle I, \beta\rangle \models (F_1 \wedge F_2)$ iff$_{\text{def}}$ $\langle I, \beta\rangle \models F_1$ and $\langle I, \beta\rangle \models F_2$. Entailment and equivalence of formulas are straightforwardly defined in terms of the satisfaction relation: A formula F_1 *entails* a formula F_2, in symbols $F_1 \models F_2$, if and only if for all interpretations $\langle I, \beta\rangle$ it holds that if $\langle I, \beta\rangle \models F_1$ then $\langle I, \beta\rangle \models F_2$. A formula F_1 is *equivalent* to a formula F_2, in symbols $F_1 \equiv F_2$, if and only if $F_1 \models F_2$ and $F_2 \models F_1$.

2 A General View on Projection and Compilation

Projection onto Literal Scopes. Projection computation is a generalization of second-order quantifier elimination which permits, so to speak, to “quantify” upon an arbitrary set of ground literals instead of just (all ground literals with) a given predicate symbol. We pursue the following formal approach to projection computation: The syntax of formulas is extended by a projection operator $\mathsf{project}(F, S)$, where F is a formula and S specifies a set of ground literals. We call a set of ground literals in the role as argument to projection a *literal scope*. The formula $\mathsf{project}(F, S)$ is called the *literal projection* of F onto S.

Projection computation then means to compute for a given formula that contains the project operator an equivalent formula without the project operator. This is analogous to second-order quantifier elimination: computing for a given formula with second-order quantifiers an equivalent formula without second-order quantifiers. Existential second-order quantification is a special case of projection, which could be defined as $\exists p\, F \stackrel{\text{def}}{=} \mathsf{project}(F, S)$, where S is the set of all ground literals with a predicate other than p. The semantics of the project operator is specified with the following clause in the definition of the satisfaction relation, which is added to the usual clauses for the classical operators:

$$\langle I, \beta\rangle \models \mathsf{project}(F, S) \text{ iff}_{\text{def}} \begin{array}{l} \text{there exists a structure } J \text{ such that} \\ \langle J, \beta\rangle \models F \text{ and } J \cap S \subseteq I. \end{array} \qquad \text{(PROJ)}$$

Some properties of projection are displayed in Table 1. Property (viii) will be discussed in the next section. For more comprehensive material see [19,20,18,13].

Linkless Formulas. Projection does not in general distribute over conjunction, as it does over disjunction, but under the precondition that the pair of the conjuncts is *linkless outside* the scope of the projection (Tab. 1. viii). This property is defined as follows, along with related properties, discussed below:

Definition 1 (Linkless). Let F, F_1, F_2 be formulas and let S be a literal scope.

(i) The pair $\langle F_1, F_2 \rangle$ is *linkless outside* S if and only if $\mathcal{L}(F_1) \cap \widetilde{\mathcal{L}(F_2)} \subseteq S \cap \widetilde{S}$.

(ii) F is *linkless outside* S if and only if each pair of conjuncts of F is linkless outside S, where the pairs of conjuncts of F are the pairs $\langle F_1, F_2 \rangle$ for each of its subformulas of the form $(F_1 \land F_2)$ and the pairs $\langle F_1, F_1 \rangle$ for each of its subformulas of the form $\forall x F_1$.

(iii) *Fully linkless* is a synonym for *linkless outside the empty set of literals.*

The term *linkless* stems from the concept of *link* as a pair of occurrences of complementary literals, each in a different conjunct of a conjunction, in the graph-based formula view of [15]. A pair of formulas is *linkless outside* a literal scope (Def. 1.i) if all ground atoms "involved in links" between the formulas (that is, are the atoms of complementary ground instances of two literals, one in each component of the pair) are contained in the literal scope, positively as well as negatively. For example, the pair of formulas $\langle p \lor q,\ \neg p \lor q \rangle$ is linkless outside the literal scope $\{+p, -p\}$. The relation is symmetric with respect to the pair components. With Def. 1.ii, the notion of *linkless* is transferred from pairs of formulas to formulas: A formula is linkless if for each of its conjunctive subformulas the pair of conjuncts is linkless, where "conjunctive subformulas" are introduced explicitly by the $\land$ operator and implicitly by universal quantification.

Linkless formulas permit to perform certain computations with low complexity. For a propositional formula that is fully linkless, satisfiability and clausal entailment can be decided in linear time [16,20]. If F is a propositional formula that is linkless outside a literal scope S, then projection computation for $\mathsf{project}(F, S)$ can be performed in linear time by simply replacing in F all literals that are not in S with $\top$. This can be justified as follows: Since F is linkless outside S, by Tab. 1.viii and vii the $\mathsf{project}$ operator can be distributed inwards immediately in front of literals L. Now, if L is in S, then by Tab. 1.v it holds that $\mathsf{project}(L, S) \equiv L$; Otherwise by Tab. 1.vi it holds that $\mathsf{project}(L, S) \equiv \top$.

The LP Relation: Generalizing Projection and Compilation. The LP relation, defined in the following, covers in different instantiations the relationships between inputs and outputs of projection computation, compilation to linkless

Table 1. Some properties of projection that hold for formulas F, F_1, F_2, literal scopes S, S_1, S_2, and universal literals L

(i) If $F_1 \models F_2$ then $\mathsf{project}(F_1, S) \models \mathsf{project}(F_2, S)$.

(ii) $F \models \mathsf{project}(F, S)$.

(iii) $\mathsf{project}(\mathsf{project}(F, S_1), S_2) \equiv \mathsf{project}(F, S_1 \cap S_2)$.

(iv) $\mathsf{project}(F, S)$ is satisfiable iff F is satisfiable.

(v) $\mathsf{project}(L, S) \equiv L$, if $\mathcal{L}(L) \subseteq S$.

(vi) $\mathsf{project}(L, S) \equiv \top$, if $\mathcal{L}(L) \cap S = \emptyset$.

(vii) $\mathsf{project}(F_1 \lor F_2, S) \equiv \mathsf{project}(F_1, S) \lor \mathsf{project}(F_2, S)$.

(viii) If $\langle F_1, F_2 \rangle$ is linkless outside S, then
$\mathsf{project}(F_1 \land F_2, S) \equiv \mathsf{project}(F_1, S) \land \mathsf{project}(F_2, S)$.

formulas, and combinations of these. The name LP suggests the combination of a syntactic requirement related to *L*inklessness with semantic conditions related to *P*rojection.

The LP relation has four arguments, where the first three represent inputs: A formula and two literal scopes, "link scope" and "projection scope", where the former is a subset of or equal to the latter. The fourth argument represents the output, a formula. The relation constrains output formulas syntactically with respect to the link scope, and semantically with respect to the input formula and the projection scope: An output formula must be *linkless outside the link scope* and be equivalent to the input formula *relative to the projection scope.*

Definition 2 (The LP Relation). For formulas F, F' and literal scopes S_l, S_p such that $S_l \subseteq S_p$ the relation LP is defined as

$$\begin{aligned} &\mathsf{LP}(F, S_l, S_p, F') \overset{\text{def}}{\equiv} \\ &\quad F' \text{ is linkless outside } S_l, && \text{(LP-L)} \\ &\quad \mathsf{project}(F', S_p) \equiv \mathsf{project}(F, S_p). && \text{(LP-P)} \end{aligned}$$

The LP relationship is preserved if the link scope S_l is enlarged, and also if the projection scope S_p is made smaller, that is, if $S_l \subseteq S'_l \subseteq S'_p \subseteq S_p$, then $\mathsf{LP}(F, S_l, S_p, F')$ implies $\mathsf{LP}(F, S'_l, S'_p, F')$.

Consider a program that computes for inputs F, S_l, S_p, where F is a propositional formula and S_l, S_p are representations of literal scopes such that $S_l \subseteq S_p$, an output formula F' such that $\mathsf{LP}(F, S_l, S_p, F')$ is satisfied. (The restriction to *propositional* F ensures that such an F' exists.) The program can be applied to solve the following different tasks, distinguished by the values of S_l and S_p:

- *Projection Computation.* If S_l and S_p are identical, then projection computation for $\mathsf{project}(F, S_p)$ can be performed by substituting in F' all literals that are not in S_p with $\top$, as indicated in Sect. 2. Thus, the essential work for projection computation is done by computing F'.
- *Compilation to an equivalent formula that is linkless outside a given literal scope.* If S_p is the set of all literals, then Condition LP-P states that F' is equivalent to F. If S_l is the given literal scope, then Condition LP-L states that F' is linkless outside the given scope.
- *Compilation to a fully linkless equivalent formula.* This specializes the task described in the previous item by requiring S_l to be the empty set. As before, S_p is the set of all literals.
- *Deciding satisfiability.* If S_l is the empty set, then F' is fully linkless and its satisfiability can be checked in a linear postprocessing step. Since projection preserves equi-satisfiability, an arbitrary set of literals can be taken as S_p. However, the empty set as S_p is preferable, since it least constrains LP.

3 LP-Tableaux

Tableau Representation. We represent tableaux as terms, constructed like formulas from literals and logic operators, but with two additional structure indicating operators. Tableau manipulation rules can then be presented as term rewriting rules which are very similar to formula rewriting rules and thus straightforwardly exhibit semantic and structural relationships between subformulas associated with subterms of tableaux.

Definition 3 (Expression, Tableau, Leafy Formula)

(i) An *expression* is constructed from literals and logic operators, like a formula, and the two additional binary operators $/\!/$ and $/$. The operators $/\!/$ and $/$ are written in infix notation, with higher precedence than $\land, \lor, \bigwedge, \bigvee$, and lower precedence than the other operators.

(ii) A *tableau* is an expression which has one of the following forms:

1. *Leaf node:* $L/\!/F$, where $L \in \mathsf{LIT} \cup \{\top\}$ and F is a sentence,
2. *Or-node:* $(L/F \land \bigvee_{i\in\{1,\ldots,n\}} T_i)$, where $n \geq 1$, $L \in \mathsf{LIT} \cup \{\top\}$, F is a sentence, and for $i \in \{1,\ldots,n\}$ it holds that T_i is a tableau.

(iii) A subexpression occurrence in a tableau is called a *subtableau* of the tableau if the subexpression is a tableau.

(iv) The *leafy formula* of an expression E, in symbols $E^{\clubsuit}$, is the NNF formula obtained from E by replacing all subexpressions of the form L/F with L and those of the form $L/\!/F$ with $(L \land F)$.

A tableau according to Def. 3.ii can be viewed as representing a tableau in the more conventional sense, an ordered tree: A *leaf node* represents a leaf of the tree. An *or-node* $(L/F \land \bigvee_{i\in\{1,\ldots,n\}} T_i)$ represents a non-leaf node, whose i-th child is the root of the tree represented by T_i. With a node $L/\!/F$ or $(L/F \land \ldots)$ two labels are associated, *literal label* L and *forward label* F. Literal labels correspond to the usual node labels in tableaux. They are restricted to literals, in contrast to complex formulas, as common for tableaux formats used by efficient automated methods. They can contain universal, but not rigid, variables. The truth value constant $\top$ is allowed as literal label to facilitate a technique discussed in Sect. 4.

If tableau node labels are restricted to literals, the – complex – input formula must be represented in some special way, external to the tableau. *Forward labels* provide a means to avoid such external structures, and, moreover, to incorporate formula simplifications and decompositions that are useful for transformation tasks but can not be straightforwardly expressed as tableau manipulations. The *forward label* of a node is a complex formula. Calculi construct it basically as a copy of the input formula on which simplifications with respect to the nodes on the branch to the node have been performed. The name *forward* should suggest that forward labels represent portions of the input that have not yet been processed (entered into the proper tableau structure), but will so at a future *"forward"* point in time. In implementations, forward labels not necessarily have to be fully materialized. A prototypical example is the DPLL procedure: The input formula, simplified by unit propagation with respect to a branch maintained by the procedure, can be modeled as forward label.

The formula view of a tableau is made explicit with the *leafy formula* operation $^{\clubsuit}$, which removes the structural operators $/$ and $/\!/$. Constituents of a leafy formula are all literal labels and the forward labels of leaf nodes. Forward

Table 2. Required properties of restriction $F|_L$ and simplification F^*

(R1) $L \wedge F \equiv L \wedge F|_L$.

(R2) $\mathcal{L}(F|_L) \subseteq \mathcal{L}(F)$.

(R3) $\mathcal{A}(F|_L) \cap \mathcal{A}(L) = \emptyset$.

(S1) $\mathsf{project}(F^*, S) \equiv \mathsf{project}(F, S)$.

(S2) $\mathcal{L}(F^*) \subseteq \mathcal{L}(F)$.

labels F of non-leaf nodes $(L/F \wedge ...)$ are not taken over into the leafy formula. They facilitate destructive tableau operations such as backjumping by recording forward labels of previous leafs. *Leafy formula* is defined for *expression* to be applicable also to tableau subexpressions which themselves are not tableaux.

Abstract Calculi. Our aim is to apply tableau construction methods to compute the LP relation. To this end, we consider tableaux that are constructed by rewrite rules. It should be noted that at this level of abstraction rules are characterized by means of semantic relationships, similar to the abstract formalization of DPLL in [17]. It is implicitly assumed that the rules are applied in cases where these relationships can be efficiently established, usually with well known methods.

We call sets of the considered rewrite rules *LP-Calculi* and the constructed tableaux *LP-tableaux.* Before we come to their definition (Def. 4), we need to introduce two mappings between formulas, which are involved in the rules.

The first is *restriction* of a sentence F by a literal sentence L, in symbols $F|_L$. Of this operation, we require that it satisfies the properties given in Tab. 2. For propositional sentences, restriction can be realized simply by replacing in F all occurrences of L with $\top$ and of $\widetilde{L}$ with $\bot$. A variant for clausal first-order sentences and universal literals is hinted in Sect. 5.

The second auxiliary operation is *scope preserving simplification* of a sentence F with respect to a literal scope S. In symbols it is written as F^*, assuming that the scope parameter is specified in the context. It is assumed to be a simplification, that is, intuitively, an operation that can be performed fast and be applied only a number of times that is polynomial in the size of the formula. The properties required of a scope preserving simplification are also listed in Tab. 2. An example that applies to clausal propositional sentences is the replacement of clauses containing an atom A, where $+A$ and $-A$ are not in S, by their resolvents upon A, in cases where the total number of clauses is reduced.

Definition 4 (LP-Calculus, LP-Tableau). Let F_0 be a sentence, S_l and S_p where $S_l \subseteq S_p$ be literal scopes, $|$ be a restriction function, and * be a simplification function preserving the scope S_p. An *LP-calculus* is a set of tableau rewrite rules which can be parameterized with these items. A tableau constructed by applying a finite number (including zero) of rewrite steps with rules of an LP-calculus to an initial tableau as specified with Init (Tab. 3) is called an *LP-tableau for* F_0 *obtained with* the LP-calculus. The parameter S_l is referred as *link scope* and S_p as *projection scope.*

Initial State and the Extend Rule. Consider Init and Extend defined in Tab. 3. Init specifies the initial state referenced in the definition of *LP-tableau.* Extend

Table 3. Initial state and [abstract] rules of LP-calculi

Init:	$\top/\!\!/F_0^*$	
Extend:	$L/\!\!/F \longrightarrow L/F \wedge \bigvee_{i\in\{1,\ldots,n\}} L_i/\!\!/F\vert^*_{L_i}$	if $\begin{cases} F \models \bigvee_{i\in\{1,\ldots,n\}} L_i, \text{ where } n \geq 1 \\ \text{For } i \in \{1,\ldots,n\}: \\ \quad L_i \in \mathsf{LIT} \text{ and } \mathcal{L}(L_i) \subseteq \mathcal{L}(F) \end{cases}$
And-Separate:	$L/\!\!/ \bigwedge_{i\in\{1,\ldots,n\}} F_i \longrightarrow L/ \bigwedge_{i\in\{1,\ldots,n\}} F_i \wedge \bigwedge_{i\in\{1,\ldots,n\}} \top/\!\!/F_i^*$	*See Sect. 4* if $\begin{cases} n \geq 2 \\ \text{For } i,j \in \{1,\ldots,n\} \text{ s.th. } i \neq j: \\ \quad \langle F_i, F_j\rangle \text{ is linkless outside } S_l \end{cases}$
True-Up:	$L/F \wedge (K/\!\!/\top \vee \bigvee_{i\in\{1,\ldots,n\}} T_i) \longrightarrow L/\!\!/\top$	*See Sect. 4* if $\begin{cases} n \geq 0 \\ \mathcal{L}(K) \cap S_p = \emptyset \end{cases}$
And-True-Up:	$L/F \wedge \bigwedge_{i\in\{1,\ldots,n\}} \top/\!\!/\top \longrightarrow L/\!\!/\top$	if $n \geq 2$
True-Below-Cut-Up:	$L/F \wedge (K/\!\!/\top \vee \widetilde{K}/\!\!/\top) \longrightarrow L/\!\!/\top$	if K is ground

is a tableau construction rule that can be considered as an "abstract rule" since it specifies semantic and structural conditions which, as shown in Sect. 5, are matched by various more specific rules, including well known tableau rules. Extend models the attachment of n successor nodes to a leaf. The *if*-preconditions state that the forward label F of the leaf entails a disjunction of n universal literals, whose literal bases are contained in that of F. For each disjunct L_i, a new node with literal label L_i is attached. Its forward label is the forward label F of the former leaf, restricted by L_i and then simplified, where the projection scope S_p (an implicit parameter) is preserved.

LP-Calculi Compute the LP Relation. That LP-calculi can be used to compute the LP relation is formally stated as Theorem 1. For clarity, we just consider instances of Extend as tableau rules, but the proofs in essence extend also to other rules, as indicated in Sect. 4. Theorem 1 refers to two properties which are defined before the theorem statement:

Definition 5 (Unlinking LP-Calculus, Terminal LP-Tableau)

(i) An LP-calculus is called *unlinking* if and only if for all LP-tableaux obtained by the calculus which have a leaf node whose forward label is not linkless outside the link scope S_l it holds that some subtableau can be rewritten with a rule of the LP-calculus.

(ii) An LP-tableau is called *terminal* with respect to an LP-calculus if and only if none of its subtableaux can be rewritten with a rule of the LP-calculus.

Theorem 1 (LP-Calculi Compute the LP Relation). *Let F_0 be a sentence, S_l, S_p be literal scopes such that $S_l \subseteq S_p$, and let CALC be an LP-calculus which* *is unlinking and whose rules are instances of* Extend*. If T is a terminal LP-tableau for F_0 that is obtained by CALC for link scope S_l and projection scope S_p, then* $\mathsf{LP}(F_0, S_l, S_p, T^{\clubsuit})$.

The theorem holds since the two conditions of LP are satisfied with respect to the given parameters, which is shown with two lemmas below. We first consider Condition LP-L, verified by Lemma 1 which is preceded by an auxiliary definition and proposition.

Definition 6 (Subformula Within a Leaf). Let T be an LP-tableau. A subformula occurrence in $T^{\clubsuit}$ is called *within a leaf with respect to T* if and only if it occurs within a subformula F that has been obtained in the mapping of T to $T^{\clubsuit}$ by replacing a leaf node $L /\!\!/ F$ with F.

Proposition 1. *Let S_l, S_p be literal scopes such that $S_l \subseteq S_p$ and assume that T is an LP-tableau obtained with an LP-calculus whose rules are instances of* Extend*, for link scope S_l and projection scope S_p. If there is an occurrence of a subformula of the form $(G \land H)$ in $T^{\clubsuit}$ that is not within a leaf with respect to T, then $\langle G, H \rangle$ is linkless outside S_l.*

By Prop. 1, a subformula $(F_1 \land F_2)$ where $\langle F_1, F_2 \rangle$ is not linkless outside S_l can in $T^{\clubsuit}$ only occur *within a leaf.* If T is terminal with respect to an unlinking calculus, this possibility is also excluded, which implies the lemma for LP-L:

Lemma 1 (Terminal LP-Tableaux are Linkless Outside the Link Scope) *Let S_l be a literal scope and let CALC be an LP-calculus which is unlinking and whose rules are instances of* Extend*, for link scope S_l. If T is a terminal LP-tableau that is obtained by CALC, then $T^{\clubsuit}$ is linkless outside S_l.*

We now turn to Condition LP-P, which is verified by Lemma 3. Its proof is based on the following Lemma 2, which states a precondition under which the result of rewriting a subformula G of F with a formula G' such that $\mathsf{project}(G', S) \equiv \mathsf{project}(G, S)$ yields a formula F' such that $\mathsf{project}(F', S) \equiv \mathsf{project}(F, S)$. Following [7], we use the following notation: If T is a term with a subterm occurrence replaced by a *hole*, and U is a term, then $T[U]$ is T with the hole replaced by U. At the same time $T[U]$ indicates that the term T contains an occurrence of the subterm U.

Lemma 2 (Scope Preservation by Unlinked Replacement). *If S is a literal scope and $G, G', F[G]$ are sentences such that (1.)* $\mathsf{project}(G', S) \equiv \mathsf{project}(G, S)$*, (2.) for all subformulas of F of the form $(F_1[G] \land F_2)$ or $(F_2 \land F_1[G])$ it holds that $\langle G, F_2 \rangle$ is linkless outside S, (3.) for all subformulas of F of the form $(F_1[G] \lor F_2)$ or $(F_2 \lor F_1[G])$ it holds that F_1 and F_2 do not have free variables in common, and (4.) conditions (2.–3.) also hold for G' in place of G, then* $\mathsf{project}(F[G'], S) \equiv \mathsf{project}(F[G], S)$.

Proof (Sketch). The sentence $F[G]$ is equivalent to a sentence $((G \land F_1) \lor F_2)$, where F_1 and F_2 are sentences and $\langle G, F_1 \rangle$ is linkless. This sentence can be obtained from $F[G]$ by rewriting subformulas of the form $((H[G] \lor H_1) \land H_2)$ – where H, H_1

and H_2 are sentences – with the equivalent $((H[G] \wedge H_2) \vee (H_1 \wedge H_2))$, and in addition some standard equivalences. Similarly, $F[G']$ is equivalent to $((G' \wedge F_1) \vee F_2)$, where $\langle G', F_1 \rangle$ is linkless too. From precondition (1.) follows $\mathsf{project}((G' \wedge F_1) \vee F_2, S) \equiv \mathsf{project}((G \wedge F_1) \vee F_2, S)$ which then implies the conclusion of the lemma. □

It can be shown that a rewriting step with Extend matches the preconditions of Lemma 2 with respect to leafy formulas: Precondition (1.) can be verified from the definition of Extend, (2.) follows as a corollary from Prop. 1 since $S_l \subseteq S_p$, and (3.) follows from the definition of tableau. By induction on the tableau structure, as constructed from Init by rewriting with Extend, the lemma for LP-P can be proven:

Lemma 3 (LP-Calculi Preserve the Projection Scope). *Let F_0 be a sentence, S_p be a literal scope, and let CALC be an LP-calculus whose rules are instances of* Extend. *If T is an LP-tableau for F_0 that is obtained by CALC for projection scope S_p, then* $\mathsf{project}(T^{\clubsuit}, S_p) \equiv \mathsf{project}(F_0, S_p)$.

4 LP-Tableau Refinements

And-Nodes. The definition of *tableau* (Def. 3.ii) can be extended by the following clause:

3. *And-node:* $(L/F \wedge \bigwedge_{i \in \{1,\ldots,n\}} T_i)$, where $n \geq 2$, $L \in \mathsf{LIT} \cup \{\top\}$, F is a sentence, and for $i \in \{1, \ldots, n\}$ it holds that T_i is a tableau.

And-nodes are like or-nodes, but with $\bigwedge$ in place of $\bigvee$ and n restricted by $n \geq 2$. By the latter restriction, tableaux with a single child are unambiguously classified as or-nodes. And-nodes are constructed with the And-Separate rule (Tab. 3). In this rule, the $\bigwedge$ operator on the left side is understood modulo associativity and commutativity. The rationale for Theorem 1 given in Sect. 3 straightforwardly carries over to LP-tableaux with and-nodes and the And-Separate rule. And-Separate constructs nodes with literal label $\top$ to preserve the structural uniformity of the tableau.

And-nodes allow to model a decomposition technique that has been introduced in DPLL adaptions for model counting [2] and also applied to knowledge compilation into DNNF[1] [11]. There, the component formulas $F_1, \ldots, F_n$ are required to have pairwise disjoint atom bases. After processing them independently, the numbers of their models are multiplied, or their compiled equivalents are conjoined, respectively. For the purpose of compilation to formulas that are linkless outside link scope S_l, the linklessness condition on $F_1, \ldots, F_n$ in And-Separate, which is weaker than requiring disjoint atom bases, is sufficient. Forward labels are quite convenient to formally handle the integration of such decomposition techniques into tableau methods.

[1] *Decomposable negation normal form (DNNF)* [4] is a sublanguage of fully linkless propositional formulas: An atom is not allowed to occur in both conjuncts of conjunctive subformulas, no matter whether in complementary or identical literals.

Tableau Level Simplifications. Simplification operations * map forward labels, or subformulas of them, to further forward labels, but they do not modify the tableau structure. Here we consider rules which modify the tableau structure, but whose effect can also be viewed as a formula simplification – of the tableau's leafy formula. Following [10], we call them *at the tableau level.*

True-Up (Tab. 3) is such a rule. The disjunction operators on its left side are understood modulo associativity and commutativity. It does in general not preserve equivalence of the leafy formula, but equivalence with respect to the projection scope S_p. As can be easily verified, if T, T' are LP-tableaux matching the two sides of an instance of True-Up, then $\mathsf{project}(T'^{\clubsuit}, S_p) \equiv \mathsf{project}(T^{\clubsuit}, S_p)$. And-True-Up (Tab. 3) supplements True-Up by applying to tableaux with a particular form that can arise when And-Separate is involved. The given rationale of Theorem 1 straightforwardly carries over to LP-tableaux whose rules include True-Up and And-True-Up.

In the case that L is – like K – not in the projection scope S_p, a rewriting step with True-Up can effect that True-Up becomes applicable again, with the old L in the role of the new K. If S_p is empty, in this way, when a model has been found – indicated by a leaf node with forward label $\top$ – the True-Up rule can be repeatedly applied until the whole tableau consists of just a single node $\top /\!\!/ \top$. The method then terminates "inherently" after finding the first model instead of having to be stopped extraneously from computing alternative models.

Tableau level simplifications have also been considered in [10], but only simplifications that preserve equivalence. Rule [10, Fig 1., rightmost] seems useful for LP-calculi, especially since it can effect that True-Up becomes applicable. Its adaption is shown in Tab. 3 as True-Below-Cut-Up.

Backjumping. Dependency directed backtracking belongs to the important techniques of efficient automated tableau systems. Backjumping, as abstractly described for DPLL in [17], is a variant of it, which can be coarsely modeled for LP-tableaux as a two-step process: An application of Extend, which is preceded by an application of the following auxiliary rule Truncate-for-Backjump:

$$L/F \wedge \bigvee_{i \in \{1,\ldots,n\}} T_i \;\longrightarrow\; L /\!\!/ F \qquad\qquad \text{if } n \geq 2$$

Backjumping is applicable to a subtableau matching the left side of Truncate-for-Backjump when it has been established (by means of data structures not represented in LP-tableaux) that $F \models K$ for a ground literal $K \in \mathcal{L}(F)$. The subtableau $L /\!\!/ F$ resulting from Truncate-for-Backjump is then rewritten by Extend to $(L/F \wedge K /\!\!/ F|^*_K)$. The given rationale of Theorem 1 can be extended to apply also for LP-tableaux with Truncate-for Backjump.

For theorem proving or model computation tasks, backjumping is commonly applied with respect to a branch in which siblings to the right represent possibilities to explore in future computation and siblings to the left are either not present or can be ignored, since they correspond to parts of the problem that already have been solved. For transformation tasks this is different. Although siblings to the left might correspond to already transformed parts of the input formula, a backjumping step can effect that these are deleted and an improved transformation, in which more unit lemmas are available below some nodes, is computed. Examples for this can be found in [20], along with a more fine grained modeling of backjumping.

The termination arguments for DPLL with backjumping (e.g. [17]) can be generalized for propositional transformation tasks. An ordering relation $>_t$ on tableaux can be defined such that $T >_t T'$ whenever T' is obtained from T by a rewrite step with one of the rules in Tab. 3 or by backjumping, as a Truncate-for-Backjump step immediately followed by Extend. The relation $>_t$ can be defined as follows: With a tableau T a sequence of pairs $\langle l_0, r_0\rangle \ldots \langle l_{n-1}, r_{n-1}\rangle$ is associated, where n is the size of its longest branch and for $i \in \{0, \ldots, n-1\}$ the left component l_i is defined as the number of leaf nodes in T at depth i which do not have $\top$ as forward label, and the right component r_i is defined as the number of nodes in T at depth $i+1$. (The root is understood as having depth 0.) Now, $T_1 >_t T_2$ if and only if the sequence associated in this way with T_1 is lexicographically greater than that associated with T_2, where the elements of the sequences are compared lexicographically, and the components of these elements by numerical value.

5 Instantiations of the LP-Tableau Framework

DPLL Adaptions. Rules Split and Unit (Tab. 4) are two instances of Extend, familiar from the DPLL procedure for propositional sentences. Split ensures that an LP-calculus has the *unlinking* property, and thus can be used to compute LP according to Theorem 1. In order to just achieve the *unlinking* property, the legitimacy of K in the precondition of Split can be further restricted by requiring $\{K, \widetilde{K}\} \not\subseteq S_l$ and that F has a subformula $(F_1[K] \land F_2[\widetilde{K}])$. On the other hand, performing Split on literals K that do not meet these criteria might be heuristically beneficial in cases where it enables effective simplifications by the * function. The Unit rule is an option. LP-calculi allow unit propagation also to be performed just within computation of forward labels by the * function.

In [5,11] an adaption of DPLL for knowledge compilation of propositional CNF into DNNF is presented, by means of pseudocode, where the compilation result is understood as trace of a procedure run. Apart from caching techniques, this method can in essence be modeled more abstractly by an LP-calculus with Split, Unit, And-Separate (for the DNNF target format, the precondition of And-Separate has to be strengthened: The F_i must have pairwise disjoint atom bases) and backjumping, supplemented by a rule application strategy that corresponds to depth-first tree construction.

DPLL and other practically successful tableau procedures operate space efficiently by keeping in memory at any point of time just a piece of the tableau under construction. While this is clearly beneficial for theorem proving tasks where a yes/no answer is searched, it can also be utilized for formula transformation tasks by methods that output pieces of their overall output as soon as they are computed. Such a method has been described for an LP-calculus (in a different notational framework) without And-Separate, but with backjumping [20].

Table 4. Instances of the Extend abstract rule

Split:	
$L /\!\!/ F \;\longrightarrow\; L/F \wedge (K /\!\!/ F\vert^*_K \vee \widetilde{K} /\!\!/ F\vert^*_{\widetilde{K}})$	if $K, \widetilde{K} \in \mathcal{L}(F)$
Unit:	
$L /\!\!/ F \;\longrightarrow\; L/F \wedge K /\!\!/ F\vert_K *$	if $\begin{cases} \mathcal{L}(K) \subseteq \mathcal{L}(F) \\ F \models K \end{cases}$
ClausalExpansion *(for propositional CNF)*:	
$L /\!\!/ F \;\longrightarrow\; L/F \wedge \bigvee_{i \in \{1,\ldots,n\}} L_i /\!\!/ F\vert^*_{L_i}$	if $\bigvee_{i \in \{1,\ldots,n\}} L_i$ is a clause in F, where $n \geq 1$
HyperTableauExtension *(for CNF)*:	
$L /\!\!/ F \;\longrightarrow\; L/F \wedge \bigvee_{i \in \{1,\ldots,n\}} {+A_i\sigma} /\!\!/ F\vert^*_{+A_i\sigma}$	if $\begin{cases} \bigvee_{i \in \{1,\ldots,n\}} {+A_i} \text{ is a clause in } F\text{, where } n \geq 1 \\ \bigvee_{i \in \{1,\ldots,n\}} {+A_i\sigma} \text{ is pure [1]} \end{cases}$

Propositional Clausal Tableaux. In [16] it has been observed that a fully developed regular clausal tableau for a given propositional CNF formula represents an equivalent to the formula that is in DNNF; thus any method for computing such a tableau can be considered as a DNNF compiler. Such a method can be modeled by an LP-calculus with an instance of Extend, the rule ClausalExpansion (Tab. 4), and a simplification * that just propagates truth value constants (removing clauses containing $\top$, removing $\bot$ from clauses, returning $\top$ for the empty clause set, and $\bot$ for a clause set containing the empty clause). In the LP-calculus framework, a clause attached by ClausalExpansion stems not directly from the input formula, but from the simplified forward label of the node to which it is attached. In this way, the violation of regularity and the construction of branches with complementary literals is automatically prevented. In a terminal tableau, each leaf has a truth value constant as forward label.

In [16] also a second method for compilation of propositional NNF formulas into DNNF has been proposed, but not related to the clausal tableau construction: Rewriting of arbitrary subformulas with the Shannon expansion. As shown in [20], this method can be simulated by LP-calculi with Split and And-Separate.

Incorporating Projection Computation into Knowledge Compilation Projection to an application relevant subvocabulary is a means to compensate somewhat for the size blow-up inherent in knowledge compilation. In the literature, so far projection computation has mainly been considered as an a-posteriori operation that can be performed with low cost on results of compilation [4,6,16] (An exception is the DPLL-based clausal compiler described in [14] which incorporates Boolean quantifier elimination). But projection computation can in part be incorporated into knowledge compilation, effecting potentially drastic efficiency improvements of the compilation process, since superpolynomial size reductions by projection become effective already in intermediate compilation stages [20, Theorem 7]. With LP-calculi this is realized by the projection scope parameter, which is passed into the interior of the calculus: The simplification function * can involve operations that preserve equivalence just with respect to the projection scope, and is applied to intermediate subformulas. The $\mathsf{True\text{-}Up}$ rule performs special cases of projection computation in intermediate stages as simplifications at the tableau level.

Hyper Tableaux. The hyper tableau calculus [1] is typically used in applications for first-order model computation where the intended semantics of a formula is the set of its minimal Herbrand models. Models computed by the hyper tableau calculus per se are not minimal. Applications accept non-minimal outputs as harmless redundancies, or use extra means to enforce minimality [3]. Hyper tableaux add two new aspects to the previously considered LP-calculi instantiations: first-order clauses and minimization.

The rule $\mathsf{HyperTableauExtension}$ (Tab. 4) is an instance of Extend that models hyper tableau construction. Its second *if*-precondition states that σ is a substitution under which the literals $+A_i$ have pairwise disjoint sets of variables. In the rule presentation quantifiers have been omitted: Free variables are assumed to be universally quantified, where the variable scope does not extend a clause. The literals $+A_i\sigma$ are understood as universal literals.

A restriction function $F|_L$ for a first-order CNF F and universal literal L (without multiple occurrences of the same variable) can be realized for example by applying $\mathsf{diff\text{-}expansion}$ [18] to compute a CNF which has the same Herbrand expansion as F (with respect to the function symbols in the fixed signature Σ) and does not contain a literal whose atom is unifiable with – but no instance of – the atom of L. In the formula resulting from diff-expansion, instances of L are then replaced by $\top$ and instances of $\widetilde{L}$ by $\bot$. For example, if the function symbols in Σ are the constant a and the unary function symbol f, then $(\forall x(-\mathsf{p}(x) \lor +\mathsf{q}(x)) \land \forall x(+\mathsf{p}(x) \lor +\mathsf{r}(x)))|_{+\mathsf{p}(\mathsf{a})} = (\mathsf{q}(\mathsf{a}) \land \forall x(-\mathsf{p}(\mathsf{f}(x)) \lor +\mathsf{q}(\mathsf{f}(x))) \land \forall x(+\mathsf{p}(\mathsf{f}(x)) \lor +\mathsf{r}(\mathsf{f}(x))))$. As a first approximation, the simplification * can be just equivalence preserving truth value propagation, as described above for clausal tableaux.

We now consider hyper tableau construction in combination with projection. Define min as logic operator such that the models of $\mathsf{min}(F)$ are exactly the minimal models of F:

$$\begin{array}{lll} \langle I, \beta\rangle \models \mathsf{min}(F) \text{ iff}_{\text{def}} & \langle I, \beta\rangle \models F \text{ and} & \text{(MIN)} \\ & \text{there does not exist a structure } J \text{ such that} & \\ & \langle J, \beta\rangle \models F \text{ and } J \cap \mathsf{POS} \subset I \cap \mathsf{POS}. & \end{array}$$

For expressions T, define $T^{\spadesuit}$ as the formula which is defined like $T^{\clubsuit}$, except that leaf nodes $L /\!\!/ F$ are replaced just by L instead of $(L \land F)$. It then holds in general that $T^{\clubsuit} \models T^{\spadesuit}$. For an LP-tableau T obtained with an LP-calculus with $\mathsf{HyperTableauExtension}$ as the only rule, $T^{\spadesuit}$ contains only positive literals and if T is terminal then $\mathsf{min}(T^{\spadesuit}) \equiv \mathsf{min}(T^{\clubsuit})$. Since $T^{\spadesuit}$ contains only positive literals it is fully linkless and $\mathsf{project}(T^{\spadesuit}, S_p)$ can be computed linearly by substituting literals of which no instance is in S_p with $\top$, as shown in Sect. 2 (assuming further that for each literal in $T^{\spadesuit}$ either all or no instances are in S_p).

Theorem 2, which follows, then justifies a method to compute $\mathsf{project}(F_0, S_p)$ where S_p contains only positive literals: Compute a terminal tableau T with $\mathsf{HyperTableauExtension}$ for F_0 and projection scope S_p and apply the linear projection computation by substitution with $\top$ to $T^{\spadesuit}$.

Theorem 2 also justifies an extension to standard hypertableau methods which is useful for projection computation and potentially also for model computation in cases where only model fragments from a subvocabulary are relevant to the application: The use of simplifications * which just preserve equivalence with respect to the projection scope. As a simple example consider the CNF sentence $F_0 = (+\mathsf{p} \land (-\mathsf{p} \lor +\mathsf{q} \lor +\mathsf{r}) \land (-\mathsf{q} \lor +\mathsf{s}) \land (-\mathsf{r} \lor +\mathsf{s}))$ and projection scope $S_p = \{+\mathsf{p}, +\mathsf{s}\}$. A value for F_0^* could then be $(+\mathsf{p} \land (-\mathsf{p} \lor +\mathsf{s}))$, which leads with one $\mathsf{HyperTableauExtension}$ step to the terminal tableau $(+\mathsf{p}/F_0^* \land +\mathsf{s}/\!/\top)$, without the need to branch for $+\mathsf{q}$ and $+\mathsf{r}$.

Theorem 2 (Projection and Hyper Tableaux). *Let F_0 be a sentence in CNF, $S_p \subseteq \mathsf{POS}$ be a literal scope and let CALC be an LP-calculus with* $\mathsf{HyperTableauExtension}$ *as its only rule. If T is a terminal LP-tableau for F_0 that is obtained by CALC for projection scope S_p, then $\mathsf{project}(T^{\spadesuit}, S_p) \equiv \mathsf{project}(F_0, S_p)$.*

Proof (Sketch). By Lemma 3 it holds that $\mathsf{project}(F_0, S_p) \equiv \mathsf{project}(T^{\clubsuit}, S_p)$. The theorem is then implied by $\mathsf{project}(T^{\clubsuit}, S_p) \equiv \mathsf{project}(T^{\spadesuit}, S_p)$, which can be shown as follows: The left to right direction is implied by $T^{\clubsuit} \models T^{\spadesuit}$ which holds in general. It remains to show the right-to-left direction. Since $T^{\spadesuit}$ contains no existential quantifiers, it satisfies the following condition: Each model of $T^{\spadesuit}$ is an extension (i.e. superset w.r.t. positive literals) of a minimal model of $T^{\spadesuit}$. From this condition and the fact that $S_p \subseteq \mathsf{POS}$ it can be derived that $\mathsf{project}(T^{\spadesuit}, S_p) \models \mathsf{project}(\mathsf{min}(T^{\spadesuit}), S_p)$. As mentioned above, since T is terminal it holds that $\mathsf{min}(T^{\spadesuit}) \equiv \mathsf{min}(T^{\clubsuit})$. It then follows that $\mathsf{project}(T^{\spadesuit}, S_p) \models \mathsf{project}(\mathsf{min}(T^{\clubsuit}), S_p)$, which implies $\mathsf{project}(T^{\spadesuit}, S_p) \models \mathsf{project}(T^{\clubsuit}, S_p)$. □

6 Conclusion

We presented an approach for applying tableau methods to projection computation, a generalization of second-order quantifier elimination. We developed a formal framework that extends, subsumes and relates a variety of methods and observations that so far have been formulated dispersed and in more ad-hoc ways. We have discussed some subtle issues exposed by the framework, such as the integration of formula simplifications that perform projection computation into knowledge compilation, and projection in combination with minimal model computation. Since the framework can be used to model techniques of efficient automated systems, it provides a basis for the specification of implementations.

References

1. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper tableaux. In: Orłowska, E., Alferes, J.J., Moniz Pereira, L. (eds.) JELIA 1996. LNCS(LNAI), vol. 1126, pp. 1–17. Springer, Heidelberg (1996)
2. Bayardo, R.J., Pehoushek, J.D.: Counting models using connected components. In: AAAI-2000, pp. 157–162 (2000)
3. Bry, F., Yahya, A.H.: Positive unit hyperresolution tableaux and their application to minimal model generation. J. Autom. Reason. 25(1), 35–82 (2000)

4. Darwiche, A.: Decomposable negation normal form. JACM 48(4), 608–647 (2001)
5. Darwiche, A.: New advances in compiling CNF to decomposable negation normal form. In: ECAI 2004, pp. 328–332 (2004)
6. Darwiche, A., Marquis, P.: A knowledge compilation map. JAIR 17, 229–264 (2002)
7. Dershowitz, N., Plaisted, D.A.: Rewriting. In: Handbook of Automated Reasoning, vol. I, ch. 1, pp. 537–610. Elsevier Science, Amsterdam (2001)
8. Ebbinghaus, H.-D., Flum, J., Thomas, W.: Einführung in die mathematische Logik, 4th edn. Spektrum Akademischer Verlag, Heidelberg (1996)
9. Gabbay, D.M., Schmidt, R.A., Szałas, A.: Second-Order Quantifier Elimination: Foundations, Computational Aspects and Applications. College Publications (2008)
10. Hähnle, R., Murray, N.V., Rosenthal, E.: Normal forms for knowledge compilation. In: Hacid, M.-S., Murray, N.V., Raś, Z.W., Tsumoto, S. (eds.) ISMIS 2005. LNCS, vol. 3488, pp. 304–313. Springer, Heidelberg (2005)
11. Huang, J., Darwiche, A.: DPLL with a trace: From SAT to knowledge compilation. In: IJCAI 2005, pp. 156–162 (2005)
12. Kohlas, J., Haenni, R., Moral, S.: Propositional information systems. J. Logic and Comp. 9(5), 651–681 (1999)
13. Lang, J., Liberatore, P., Marquis, P.: Propositional independence – formula-variable independence and forgetting. JAIR 18, 391–443 (2003)
14. McMillan, K.L.: Applying SAT methods in unbounded symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 250–264. Springer, Heidelberg (2002)
15. Murray, N.V., Rosenthal, E.: Dissolution: Making paths vanish. JACM 40(3), 504–535 (1993)
16. Murray, N.V., Rosenthal, E.: Tableaux, path dissolution and decomposable negation normal form for knowledge compilation. In: Cialdea Mayer, M., Pirri, F. (eds.) TABLEAUX 2003. LNCS, vol. 2796, pp. 165–180. Springer, Heidelberg (2003)
17. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). JACM 53(6), 937–977 (2006)
18. Wernhard, C.: Semantic knowledge partitioning. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 552–564. Springer, Heidelberg (2004)
19. Wernhard, C.: Literal projection for first-order logic. In: Hölldobler, S., Lutz, C., Wansing, H. (eds.) JELIA 2008. LNCS (LNAI), vol. 5293, pp. 389–402. Springer, Heidelberg (2008)
20. Wernhard, C.: Automated Deduction for Projection Elimination. Dissertationen zur Künstlichen Intelligenz (DISKI), vol. 324. AKA/IOS Press (2009)

Author Index