Lecture Notes in Artificial Intelligence 4548

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Nicola Olivetti (Ed.)

Automated Reasoning with Analytic Tableaux and Related Methods

16th International Conference, TABLEAUX 2007
Aix en Provence, France, July 3-6, 2007
Proceedings

Springer

Series Editors

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

Volume Editor

Nicola Olivetti
LSIS - UMR CNRS 6168
Université Paul Cézanne
Domaine Universitaire de Saint-Jérôme
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20, France
E-mail: nicola.olivetti@univ-cezanne.fr

Library of Congress Control Number: 2007929030

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 3-540-73098-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-73098-9 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12077922 06/3180 5 4 3 2 1 0

Preface

This volume gathers the research papers presented at the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2007) that took place July 3–6, 2007 in Aix en Provence, France. This conference was the 16th in a series of international meetings held since 1992 (the list is on page VIII).

The Program Committee of TABLEAUX 2007 received 43 submissions, 16 of which were accepted for publication in the present proceedings, while 8 were accepted as position papers.

In addition to the contributed papers, the program included three excellent keynote talks by Piero Bonatti of Università di Napoli, by John-Jules Meyer of Utrecht University, and by Cesare Tinelli of the University of Iowa. Finally, the program was completed by three tutorials of deep interest: “The Tableau Work Bench: Theory and Practice” (Pietro Abate and Rajeev Goré), “Tableau Methods for Interval Temporal Logics” (Valentin Goranko and Angelo Montanari), and “Semistructured Databases and Modal Logic” (Serenella Cerrito).

Tableaux and related methods are a convenient formalism for automating deduction in classical as well as in non-classical logics. The papers collected in this volume witness the wide range of logics being covered: from intuitionistic and substructural logics to modal logics (including temporal and dynamic logics), from many-valued logics to nonmonotonic logics, from classical first-order logic to description logics. Some contributions are focused on decision procedures, others on efficient reasoning, as well as on implementation of theorem provers. A few papers explore applications such as model-checking, verification, or knowledge engineering. Finally, some contributions make use of tableaux as a tool for theoretical investigation of logics. This variety of logics and applications illustrates well the flexibility and the ubiquity of analytic tableaux and related proof methods.

I want to express my gratitude to the invited speakers and to the tutorial presenters who really contributed to making a rich and stimulating conference program. I am very grateful to the members of the Program Committee for their assistance in all phases of the conference and to other reviewers who ensured, together with the Program Committee members, a rigorous selection of the papers. Their effort was decisive to keeping the high scientific standard of the conference. I am also grateful to the members of the Steering Committee for their valuable support. I particularly thank my colleagues of the Local Organizing Committee: Belaid Benhamou, Djamal Habet, Philippe Jégou, Richard Ostrowski, Cyril Pain-Barre, Odile Papini, Nicolas Prcovic, Vincent Risch, Pierre Siegel, Cyril Terrioux, Eric Würbel; they worked hard with a truly cooperative spirit, making the conference a successful event. A final thanks to the Office of Tourism of Aix en Provence and to Promo Sciences for their professional assistance and services.

July 2007 Nicola Olivetti

Organization

Program and Conference Chair

Nicola Olivetti	Paul Cézanne University, France

Program Committee

Peter Baumgartner	NICTA, Australia
Bernhard Beckert	University of Koblenz-Landau, Germany
Patrick Blackburn	INRIA Lorraine, France
Marta Cialdea	University of Roma 3, Italy
Roy Dyckhoff	University of St. Andrews, UK
Christian G. Fermüller	University of Vienna, Austria
Ulrich Furbach	University of Koblenz-Landau, Germany
Didier Galmiche	LORIA, Henri Poincaré University, France
Martin Giese	RISC, Johannes Kepler University, Austria
Rajeev Goré	Australian National University, Australia
Jean Goubault-Larrecq	LSV, ENS Cachan, France
Reiner Hähnle	University of Chalmers, Sweden
Ullrich Hustadt	University of Liverpool, UK
Christoph Kreitz	University of Potsdam, Germany
Carsten Lutz	University of Dresden, Germany
Angelo Montanari	University of Udine, Italy
Ugo Moscato	University of Milano-Bicocca, Italy
Neil V. Murray	ILS Institute, University at Albany, USA
Ilkka Niemelä	Helsinki University of Technology, Finland
Lawrence C. Paulson	University of Cambridge, UK
Camilla Schwind	LIF-CNRS, France
Viorica Sofronie-Stokkermans	Max-Planck-Institut für Informatik, Germany
Arild Waaler	University of Oslo, Norway

Additional Referees

Alessandro Avellone
Matthias Baaz
Thomas Bolander
Mirjana Borisavljević
Torben Braüner
Davide Bresolin
Elie Bursztein
Serenella Cerrito
Marcello D'Agostino
Stéphane Demri
Hans de Nivelle
Camillo Fiorentini
Guido Fiorino
Lev Gordeev
Andrew Haas
Keijo Heljanko
Jan Hladik
Arnold Holger
Jinbo Huang
Yevgeny Kazakov
Kentaro Kikuchi
Dominique Larchey-Wendling
Patrick Maier

Daniel Méry
George Metcalfe
Maja Milicic
Andrea Orlandini
Mario Ornaghi
Gabriele Puppis
Robert Rothenberg
Gernot Salzer
Peter Schotch
Yaroslav Shramko
Evgeny Zolin

Steering Committee

Rajeev Goré (President)	Australian National University, Australia
Bernhard Beckert (Vice-President)	University of Koblenz-Landau, Germany
Stéphane Demri	ENS Cachan, France
Roy Dyckhoff	University of St. Andrews, UK
Ulrich Furbach	University of Koblenz-Landau, Germany
Didier Galmiche	LORIA, Henri Poincaré University, France
Reiner Hähnle	University of Chalmers, Sweden
Nicola Olivetti	Paul Cézanne University, France

Organization

TABLEAUX 2007 was organized by the INCA team (Inference, Constraints and Applications) of LSIS CNRS UMR 6168 (Information and System Sciences Laboratory).

Sponsoring Institutions

LSIS Laboratory CNRS UMR 6168
Université Paul Cézanne
Université de Provence
Université de la Méditerranée
Ville d'Aix en Provence
Communauté du Pays d'Aix
Conseil Général des Bouches du Rhône
Région Provence Alpes Côte d'Azur

Previous Conferences

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)

Table of Contents

Invited Talks

Research Papers

System Descriptions

Nonmonotonic Description Logics – Requirements, Theory, and Implementations

Piero A. Bonatti

Dipartimento di Scienze Fisiche
Sezione di Informatica
Università di Napoli Federico II
Napoli, Italy
bonatti@na.infn.it

Abstract. The Semantic Web and a number of modern knowledge-based applications have turned ontologies into a familiar and popular ICT notion. Description Logics (DLs) are one of the major formalisms for encoding ontologies.

Many "users" of such formalisms – that is, ontologies writers – would appreciate DLs to have nonmonotonic features. For example, it would be appealing to describe taxonomies by means of general default properties that may be later overridden in special cases; a similar behavior is supported by all object-oriented languages, after all. However, nonmonotonic extensions of DLs involve many tricky technical problems.

This talk will briefly illustrate some of the major requirements for nonmonotonic description logics and some of the formalisms currently available. Then we shall point out the major problems that still have to be solved in order to apply standard tableaux optimization techniques to nonmonotonic DLs. Since DLs are usually at least PSPACE-hard, such optimization techniques are crucial in making these formalisms usable in practice.

For example, it seems very difficult to find a tableaux system for a fragment of nonmonotonic DLs where a tableau needs not be stored entirely in memory (because it is enough to construct and verify a single branch at each iteration, for example).

Since "traditional" nonmonotonic semantics are not completely satisfactory, it may be possible to solve both semantic shortcomings and optimization problems by adopting suitable new logics.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, p. 1, 2007.

Our Quest for the Holy Grail of Agent Verification

John-Jules Ch. Meyer

Department of Information and Computing Sciences,
Utrecht University,
The Netherlands
jj@cs.uu.nl

Abstract. Since the inception of agent technology almost two decades ago, researchers have worked both on the formal, theoretical aspects of intelligent agents and on the realisation / implementation of them. However, the link between the two has always remained rather unclear, to this day. Although there is a definite need for the verification of agents, the methods and techniques for this are still in their infancy. We describe our personal ongoing quest for the 'right' approach to agent verification.

1 Cognitive Agent Programming

The basic ideas of agent programming go back to Aristotle's *practical reasoning* (cf. [5], p. 728), augmented by the modern philosophical concepts of the *intentional stance* by Dennett [16] and that of *intention* by Bratman [10]. In short, practical reasoning is about specifying the decision of an agent by coining it in a rule (a so-called practical syllogism). Together with the idea of treating an entity as a rational agent deliberating its beliefs and goals in order to come up with the next action (Dennett's intentional stance), and the idea that resource-bounded agents should always settle on some of their desires and then stick with these as long as is rationally possible (the concept of an intention), this provides the ingredients of the current agent-oriented programming languages such as Agent0 [42], AgentSpeak(L) [34] and 3APL [24,14]/ 2APL [13] (also cf. [7]). In particular, we are interested in what we call cognitive or BDI agent programming, in which the agent has mental attitudes such as beliefs, desires (goals) and intentions (plans).

For instance, in 3APL and 2APL an agent has a belief base, a goal base and a plan base. The programmer can use Plan Generation (PG) and Plan Revision (PR) rules of the form

$$\gamma|\beta \to \pi$$

and

$$\pi_1|\beta \to \pi_2$$

respectively, to let the agent generate plans π, given the agent's beliefs β and goals γ, and revise plans π_1 to π_2 when necessary (indicated by a certain belief condition β) (cf.[7,13]). As one can see, these rules are a direct operationalization of practical reasoning together with the ideas of Dennett and Bratman.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 2–9, 2007.

2 Agent Verification

Since the inception of agent technology almost two decades ago, researchers have worked both on the formal, theoretical aspects of intelligent agents and on the realisation / implementation of them [12,35,28,43]. However, the link between the two has always remained rather unclear, to this day. A case in point is the pioneering work of Rao & Georgeff, who developed their famous BDI (beliefs-desires-intentions) logic ([35]) on the one hand, while working on their BDI architecture and programming language AgentSpeak(L) ([34]) to realize agents, on the other. Although the names of the logic and architecture (both containing 'BDI') suggest an intrinsic connection, they are only superficially related. This also holds for the BDI-like logic and the programming language Agent0 in the pioneering first paper on agent-oriented programming by Shoham [42]. More generally, this phenomenon is referred to in the literature as the 'gap' between agent logics and implemented agent systems. One of the problems is that the BDI notions in agent logics generally are not 'grounded' in agent computations: they are rather general notions, modelled by abstract (accessibility) relations in modal logic, and have no apparent association with concrete agent behaviour, and an agent program in particular (cf. [44]). On the other hand one has increasingly come to realize that complex agent-based systems need verification. For instance, in the domain of space exploration one is getting more and more interested in agent systems, but also feels the need for verifying them [36]. Although here it is an absolute necessity in view of huge investments of money and the fact that possibly also human lives are at stake, there are many other examples, where the sheer complexity of agent-based systems to be deployed calls for formal verification. However, since it is hard to connect the practice of agent programming with the formal counterpart of agent logics, it has been hard to establish a solid framework for agent verification.

3 Our Approaches

In the last 15 years or so we have been working on both agent logics (viz. the KARO framework [28]) and agent programming (viz. the agent languages 3APL [24,14] and, more recently, 2APL [13]), and personally I have been always fascinated by the relation between the two, and in particular how agent logics can be used to specify and verify agent programs. Furthermore, having our 'roots' in semantics and correctness of 'traditional' programming in the style of De Bakker [6], when attempting to prove agent programs correct, we have always been very interested to (re-)use methods and techniques from 'traditional' program correctness, and try to adapt and extend them to deal with agent programs. Of course, these adaptations and extensions are needed to cater for the typical agent-oriented features such as the BDI attitudes in particular, not present and dealt with in traditional approaches.

As we saw before one of the biggest problems is to connect ('ground') BDI notions in agent logics with agent programming, i.e. more computational notions.

Agent logics are generally not grounded: notions like beliefs, goals, etc. and, more generally, possible worlds and accessibility relations are very abstract and not connected directly to computational notions. Therefore no direct link between agent logics and agent programming exists, even if both use what seems to be at first sight the same notions (such as beliefs and goals).

In this section we will discuss a number of routes we have taken attempting to bridge the gap between agent programming and logic.

3.1 Programming KARO Agents

KARO is an expressive formalism to specify agent attitudes ([28]). It is a blend of dynamic and epistemic logic augmented with modalities for motivational attitudes (desires, goals, commitments, ...) Because of the fact that it is based on a logic of *action* rather than *time*, like the other well-known agent logics, we always had the idea that KARO is closer than other agent logics to a computational approach to agents such as agent programming. In [32] we have explored the idea of putting agent programs as actions into the dynamic logic operators to be able to reason about these programs within the KARO framework, but the idea was only elaborated in a rather loose and preliminary fashion. We will see that this idea was later picked up again in a more rigorous manner.

3.2 An Executable Core of KARO

One way, of course, to establish a relation between a program and its logical specification is to use the specification as program itself. This has been the philosophy of METATEM [21]: to execute specifications of agents written in an executable fragment of temporal logic augmented with BDI concepts (see e.g. [22]). Together with the METATEM team we have looked whether the same line could be followed with specifications in the KARO framework, a blend of dynamic and epistemic logic, augmented with other BDI-like concepts (see [28]). We succeeded in mimicking this idea for KARO [29,30], but the concession we had to make is that we had to reduce the very expressive KARO framework to a rather small core, with the drawback that much of the expressiveness was lost.

3.3 The GOAL Method

In [25,23] we considered a very simple agent programming logic with declarative goals (but without intentions or procedural goals / plans!), and tried to give a complete programming theory for it, viz. a programming language together with a formal semantics, as well as a correctness logic. Interestingly, apart from the familiar Hoare triples also ideas from concurrency theory such as the UNITY framework ([11]) were used. It was meant as a kind of proof of concept that one could try to get such a complete theory for agent programming, but, admittedly, the power of the programming language, called GOAL, was rather limited (although deliberately so). Some of these ideas were later used to enhance our programming language 3APL with both declarative and procedural goals [38,14].

3.4 Agent Logics as Program Logics: Grounding BDI-Like Logics

Recently we have taken up our efforts to ground agent logics in a more principled way.

Grounding KARO. In [26,27] this is attempted by making the notions that occur in agent logics (e.g. KARO) such as beliefs and goals less abstract and more computational (e.g. by not basing them on abstract accessibility relations but on certain types of knowledge bases, yielding a 'state-based semantics'), so that reasoning about these notions becomes relevant for reasoning about concrete agent programs. But also this comes with the price of reducing the logic (KARO) to a core.

Relating accessibility and execution ($\boldsymbol{CTL_{APL}}$). In [15] it is proposed to base the abstract *accessibility relations* of agent logics directly on the (Plotkin-style) operational semantics of agent programs. In this particular proposal the temporal logic CTL* [18] is employed, where the temporal accessibility relation is specified by a transition system for the operational semantics of the agent programming language (APL) at hand, thus establishing the grounding of the logic. By doing this one can use the logical language to express properties of (the behavior of) agent programs written in APL, which may then be verified by model-checking, for instance.

A dedicated PDL version for APLs. In a recent paper [4] we go about in a different way: we consider a (simple) agent programming language SimpleAPL together with its operational semantics. We next devise a PDL-based [20]) agent logic tailored to constructs that are present in SimpleAPL. This is done by employing a function that transforms the basic ingredients of SimpleAPL to expressions in the logic. Next we give a sound and complete axiomatization of this logic. We then show the relevance of the logic for proving properties of programs written in SimpleAPL by proving a theorem exactly relating the transition semantics of the programs appearing in the PDL-like logic and the operational semantics of SimpleAPL. In this way we know that we can use the devised logic for reasoning about SimpleAPL programs, and thus show correctness properties of these programs. We show how this can be done by way of an example. The reasoning can be assisted by automated verfication methods, and actually these (viz. [41]) *were* used in verifying the example mentioned.

3.5 Dynamic Logic for 3APL

We have also proposed a verification logic for the language 3APL directly as for any other traditional programming language without resorting to a connection with agent logics. In [40,37] we propose a dynamic logic for the execution of plans where also it is taken into account that plans may be revised by plan revision (PR) rules. As one may appreciate, the latter renders the execution of plans highly 'non-compositional', which results in the fact that in this situation the standard validity in dynamic logic, $[\pi_1;\pi_2]\varphi \leftrightarrow [\pi_1][\pi_2]\varphi$, is not valid anymore.

Therefore, first a logic is given for restricted application of plan revision (at most n times, for a natural number n). Then this is extended to a logic for arbitrary plan revision, yielding an infinitary (so-called ω-)axiomatization, which is shown to be complete. Although the result is satisfying from a technical view, it is thus quite unsatisfactory in a practical sense, since it does not really yield a feasible verification method. It also raised the discussion among ourselves what exactly the program is in a language such as 3APL: the plans that are executed (and revised on the fly!), or the interpreter implementing the deliberation process (in this case the alternation of plan execution and plan revision). If we would say that it is the latter, we would rather have to look at a program (e.g. dynamic) logic where the higher-level ('meta'-) actions execute_plan and apply_rule(r) are the actions that should be reasoned about. This matter is not yet fully resolved in our minds...

3.6 Rapid Prototyping of APLs in Maude

One of our recent papers constitutes a proposal to use the term rewriting language Maude [33] for rapid prototyping agent programming languages (APLs). The idea is to write the operational semantics of an APL (in terms of a Plotkin-style transition system) directly in Maude. As Maude also comes with an LTL model checker this also provides possibilities to check properties of agent programs written in the APL at hand, that is, potentially, but as yet we have not yet pursued this line of research. We think this will be fruitful, and would bring us also closer to related work done on model-checking multi-agent systems and the agent language AgentSpeak/Jason in particular ([8,9,31]).

3.7 The 'Macro' Level of Multi-agent Systems

Here we mention some work we've done on the 'macro' level of multi-agent systems. Besides programming individual cognitive agents, it is also very important to consider systems at the larger scale of multi-agent systems or agent societies. Here we have to deal with the interesting but difficult issue of balancing the autonomy of the individual agents with a desired form of group behavior, which can be captured in norms (cf. [17]). This could be called the problem of establishing the micro-macro link in multi-agent systems. To keep the agents in line, so to speak, special software systems, called electronic institutions, may be used that regulate the agents' behavior enforcing the norms by protocols ([19]). Of course, it is then crucial to know whether these protocols (concrete procedural constraints) are in line with the norms (high-level declarative constraints). We have worked on methods and techniques adopted and adapted from traditional programming, such as temporal logic, to be able to verify that these protocols comply with the norms ([3,1]). The interesting thing here is that using these methods especially the more informal assumptions under which the protocol can be proven norm-compliant, become explicit. Furthermore, in [2,1] we have given a method to derive protocols on the basis of more declarative 'landmarks', using Büchi automata and temporal logic.

4 Conclusion

From the above it may be clear that the last word is not yet said about this topic. As can be seen from our proposals, we try to bridge the gap from both sides: from agent programming to logic, as well as from logic to programs. We feel that we do make some progress, and at least get more understanding of the issues involved.

Acknowledgments. I would like to thank my co-authors of the various referenced papers, companions on the quest for agent verification, for the many discussions on the issues raised in this paper.

References

1. Aldewereld, H.: Autonomy vs. Conformity: An Institutional Perspective on Norms and Ptotocols, Ph.D. thesis, Utrecht University, Utrecht (2007)
2. Aldewereld, H., Dignum, F., Meyer, J.-J.Ch.: Designimg Protocols for Agent Institutions, accepted for ProMAS2007 (2007)
3. Aldewereld, H., Vázquez-Salceda, J., Dignum, F., Meyer, J.-J.Ch.: Verifying Norm Compliancy of Protocols. In: Boissier, O., Padget, J., Dignum, V., Lindemann, G., Matson, E., Ossowski, S., Sichman, J.S., Vázquez-Salceda, J. (eds.) Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems. LNCS (LNAI), vol. 3913, pp. 231–245. Springer, Berlin, Heidelberg (2006)
4. Alechina, N., Dastani, M., Logan, B., Meyer, J.-J.Ch.: A Logic of Agent Programs. In: Proc. AAAI-07 (to appear, 2007)
5. Audi, R. (ed.): The Cambridge Dictionary of Philosophy. Cambridge Univ. Press, Cambridge (1999)
6. de Bakker, J.W.: Mathematical Theory of Program Correctness. Prentice-Hall International, London (1980)
7. Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.): Multi-Agent Programming. Kluwer, Boston, Dordrecht, London (2005)
8. Bordini, R.H., Moreira, A.F.: Proving the Asymmetry Thesis Principles for a BDI Agent-Oriented Programming Language, Electronic Notes in Theoretical Computer Science 70(5) (2002)
9. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Verifying multi-agent programs by model checking. Autonomous Agents and Multi-Agent Systems 12(2), 239–256 (2006)
10. Bratman, M.E.: Intentions, Plans, and Practical Reason. Harvard University Press, Massachusetts (1987)
11. Chandy, K.M., Misra, J.: Parallel Program Design. Addison-Wesley, London (1988)
12. Cohen, P.R., Levesque, H.J.: Intention is Choice with Commitment. Artificial Intelligence 42(3), 213–261 (1990)
13. Dastani, M.: Meyer, J.-J.Ch.: A Practical Agent Programming Language, accepted for ProMAS07 (2007)
14. Dastani, M., van Riemsdijk, M.B., Dignum, F., Meyer, J.-J.Ch.: A Programming Language for Cognitive Agents: Goal-Directed 3APL. In: Dastani, M., Dix, J., El Fallah-Seghrouchni, A. (eds.) PROMAS 2003. LNCS (LNAI), vol. 3067, Springer, Berlin (2004)

15. Dastani, M., van Riemsdijk, B., Meyer, J.-J.Ch.: A Grounded Specification Language for Agent Programs. In: Proc. AAMAS07, ACM Press, New York (2007)
16. Dennett, D.: The Intentional Stance, Bradford Books/MIT Press, Cambridge MA (1987)
17. Dignum, V.: A Model for Organizational Interaction (Based on Agents, Founded in Logic), Ph.D. Thesis, Utrecht University, Utrecht (2004)
18. Emerson, E.A., Halpern, J.Y.: Sometimes and Not Never Revisited: on Branching versus Linear Time Temporal Logic. J. ACM 33(1), 151–178 (1986)
19. Esteva, M., Padget, J., Sierra, C.: Formalizing a Language for Institutions and Norms. In: Meyer, J.-J.Ch., Tambe, M. (eds.) ATAL 2001. LNCS (LNAI), vol. 2333, pp. 348–366. Springer, Berlin (2002)
20. Fischer, M.J., Ladner, R.E.: Propositional Dynamic Logic of Regular Programs. J. Comput. Syst. Sci 18(2), 194–211 (1979)
21. Fisher, M.: A Survey of Concurrent METATEM – The language and Its Applications. In: Gabbay, D.M., Ohlbach, H.J. (eds.) ICTL 1994. LNCS, vol. 827, pp. 480–505. Springer, Berlin (1994)
22. Fisher, M.: Implementing Temporal Logics: Tools for Execution and Proof (Tutorial Paper). In: Toni, F., Torroni, P. (eds.) Computational Logic in Multi-Agent Systems. LNCS (LNAI), vol. 3900, pp. 129–142. Springer, Heidelberg (2006)
23. Hindriks, K.V.: Agent Programming Languages: Programming with Mental Models, Ph.D. thesis, Utrecht University, Utrecht (2001)
24. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.Ch.: Agent Programming in 3APL. Int. J. of Autonomous Agents and Multi-Agent Systems 2(4), 357–401 (1999)
25. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.Ch.: Agent Programming with Declarative Goals. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, pp. 228–243. Springer, Heidelberg (2001)
26. Hindriks, K.V., Meyer, J.-J.Ch.: An Agent Program Logic with Declarative Goals. In: Dunin-Keplicz, B., Verbrugge, R. (eds.) Proc. FAMAS06 (ECAI, Workshop on Formal Aspects of Multi-Agent Systems) ECCAI, 2006, pp. 1-15 (2006)
27. Hindriks, K.V., Meyer, J.-J.Ch.: Agent Logics as Program Logics: Grounding KARO. In: Freksa, C., Kohlhase, M., Schill, K. (eds.) KI 2006. LNCS (LNAI), vol. 4314, pp. 404–418. Springer, Heidelberg (2007)
28. van der Hoek, W., van Linder, B., Meyer, J.-J.Ch.: An Integrated Modal Approach to Rational Agents. In: Wooldridge, M., Rao, A. (eds.) Foundations of Rational Agency. Applied Logic Series, vol. 14, pp. 133–168. Kluwer, Dordrecht (1998)
29. Hustadt, U., Dixon, C., Schmidt, R.A., Fisher, M., Meyer, J.-J.Ch.: Reasoning about Agents in the KARO Framework. In: Bettini, C., Montanari, A. (eds.) Cividale del Friuli, Italy. Eighth International Symposium (TIME-01), Cividale del Friuli, Italy, pp. 206–213. IEEE Press, Los Alamitos, CA, USA (2001)
30. Hustadt, U., Dixon, C., Schmidt, R.A., Fisher, M., Meyer, J.-J.Ch.: Verification within the KARO Agent Theory. In: Rouff, C., Hinchey, M., Rash, J., Truszkowski, W., Gordon-Spears, D. (eds.) Agent Technology from a Formal Perspective, NASA Monographs in Systems and Software Engineering Series, pp. 193–225. Springer, Berlin (2006)
31. Lomuscio, A., Raimondi, F.: Mcmas: A Model Checker for Multi-Agent Systems. In: Proc. TACAS '06, 450–454 (2006)
32. Meyer, J.-J.Ch., de Boer, F.S., van Eijk, R.M., Hindriks, K.V., van der Hoek, W.: On Programming KARO Agents. Logic Journal of the IGPL 9(2), 245–256 (2001)
33. Ölveczky, P.C.: Formal Modeling and Analysis of Distributed Systems in Maude, lecture notes (2005)

34. Rao, A.S.: AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Berlin (1996)
35. Rao, A.S., Georgeff, M.P.: Modeling Rational Agents within a BDI-Architecture. In: Allen, J., Fikes, R., Sandewall, E. (eds.) Principles of Knowledge Representation and Reasoning. Second International Conference (KR'91), pp. 473–484. Morgan Kaufmann, Washington (1991)
36. Rash, J.L., Rouff, C.A., Truszkowski, W., Gordon, D.F., Hinchey, M.G. (eds.): FAABS 2000. LNCS (LNAI), vol. 1871. Springer, Berlin, Heidelberg (2001)
37. van Riemsdijk, M.B.: Cognitive Agent Programming: A Semantic Approach, Ph.D. Thesis, Utrecht University, Utrecht (2006)
38. van Riemsdijk, M.B., van der Hoek, W., Meyer, J.-J.Ch.: Agent Programming in Dribble: from Beliefs to Goals Using Plans. In: Rosenschein, J.S., Sandholm, T., Wooldridge, M., Yokoo, M. (eds.) Autonomous Agents and Multiagent Systems. 2nd Int. J. Conf (AAMASO03), Melbourne Australia, pp. 393–400. ACM Press, New York (2003)
39. van Riemsdijk, M.B., de Boer, F.S., Dastani, M., Meyer, J.-J.Ch.: Prototyping 3APL in the Maude Term Rewriting Language. In: Inoue, K., Satoh, K., Toni, F. (eds.) Computational Logic in Multi-Agent Systems. LNCS (LNAI), vol. 4371, pp. 95–114. Springer, Berlin (2007)
40. van Riemsdijk, M.B., de Boer, F.S., Meyer, J.-J.Ch.: Dynamic Logic for Plan Revision in Intelligent Agents. J Logic Computation 16(3), 375–402 (2006)
41. Schmidt, R.A.: PDL-TABLEAU, (2003) http://www.cs.man.ac.uk/schmidt/pdl-tableau
42. Shoham, Y.: Agent-Oriented Programming. Artificial Intelligence 60(1), 51–92 (1993)
43. Wooldridge, M.J.: Reasoning about Rational Agents. MIT Press, Cambridge, MA (2000)
44. van der Hoek, W., Wooldridge, M.: Towards a Logic of Rational Agency. Logic Journal of the IGPL 11(2), 133–157 (2003)

This page intentionally blank

An Abstract Framework for Satisfiability Modulo Theories

Cesare Tinelli*

Department of Computer Science
The University of Iowa
tinelli@cs.uiowa.edu

Abstract. Satisfiability Modulo Theories (SMT) studies methods for checking the satisfiability (or, dually, the validity) of first-order formulas with respect to some logical theory T of interest. What distinguishes SMT from general automated deduction is that the background theory T need not be finitely or even first-order axiomatizable, and that specialized inference methods are used for each theory. By being theory-specific and restricting their language to certain classes of formulas (such as, typically but not exclusively, ground formulas), these specialized methods can be implemented into solvers that are more efficient in practice than general-purpose theorem provers. While SMT techniques have been traditionally used to support deductive software verification, they are now finding applications in other areas of computer science such as, for instance, planning, model checking and automated test generation.

Theory-specific solvers can be often described conveniently in terms of tableau calculi, especially if one wants to prove that a solver decides a certain fragment of a theory T. In practice, however, most modern SMT solvers are not tableau-based and follow one of two main approaches, both of which exploit the recent technological advances in SAT solving. The "eager" approach uses smart encodings to propositional logic to compile T-satisfiability problems into propositional satisfiability problems, which can then be solved by off-the-self SAT solvers. The "lazy" approach instead uses general run-time mechanisms to separate plain Boolean reasoning from theory reasoning proper, doing the latter with *small* specialized procedures, and delegating the former to a dedicated SAT engine based on the DPLL procedure.

After a brief overview of SMT, this talk focuses on a general and extensible abstract framework, Abstract DPLL Modulo Theories, for modeling lazy STM solvers declaratively and studying some of their theoretical properties. The framework is used to present and discuss a few basic variants of the lazy approach, in the case of a single and of multiple background theories. The talk also presents an extension of the framework that drastically simplifies the implementation of theory-specific components, and could be of interest to implementors of ground tableaux calculi as well.

* The author's research described in this talk is the result of past and on-going collaborations with Clark Barrett, Robert Nieuwenhuis, and Albert Oliveras on the subject, and was made possible with the partial support of grants #0237422 and #0551646 from the National Science Foundation.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, p. 10, 2007.

Axiom Pinpointing in General Tableaux

Franz Baader[1] and Rafael Peñaloza[2,*]

[1] Theoretical Computer Science, TU Dresden, Germany
baader@inf.tu-dresden.de
[2] Intelligent Systems, University of Leipzig, Germany
penaloza@informatik.uni-leipzig.de

Abstract. Axiom pinpointing has been introduced in description logics (DLs) to help the user to understand the reasons why consequences hold and to remove unwanted consequences by computing minimal (maximal) subsets of the knowledge base that have (do not have) the consequence in question. The pinpointing algorithms described in the DL literature are obtained as extensions of the standard tableau-based reasoning algorithms for computing consequences from DL knowledge bases. Although these extensions are based on similar ideas, they are all introduced for a particular tableau-based algorithm for a particular DL.

The purpose of this paper is to develop a general approach for extending a tableau-based algorithm to a pinpointing algorithm. This approach is based on a general definition of "tableaux algorithms," which captures many of the known tableau-based algorithms employed in DLs, but also other kinds of reasoning procedures.

1 Introduction

Description logics (DLs) [2] are a successful family of logic-based knowledge representation formalisms, which can be used to represent the conceptual knowledge of an application domain in a structured and formally well-understood way. They are employed in various application domains, such as natural language processing, configuration, databases, and bio-medical ontologies, but their most notable success so far is the adoption of the DL-based language OWL [13] as standard ontology language for the semantic web. As a consequence of this standardization, several ontology editors support OWL [15, 18, 14], and ontologies written in OWL are employed in more and more applications. As the size of such ontologies grows, tools that support improving the quality of large DL-based ontologies become more important. Standard DL reasoners [12, 10, 24] employ tableau-based algorithms [6], which can be used to detect inconsistencies and to infer other implicit consequences, such as subsumption relationships between concepts or instance relationships between individuals and concepts.

For a developer or user of a DL-based ontology, it is often quite hard to understand why a certain consequence holds,[1] and even harder to decide how to

[*] Funded by the German Research Foundation (DFG) under grant GRK 446.

[1] Note that this consequence may also be the inconsistency of the knowledge base or the unsatisfiability of a concept w.r.t. the knowledge base.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 11–27, 2007.

change the ontology in case the consequence is unwanted. For example, in the current version of the medical ontology SNOMED [25], the concept *Amputation-of-Finger* is classified as a subconcept of *Amputation-of-Arm*. Finding the axioms that are responsible for this among the more than 350,000 terminological axioms of SNOMED without support by an automated reasoning tool is not easy.

As a first step towards providing such support, Schlobach and Cornet [22] describe an algorithm for computing all the *minimal subsets* of a given knowledge base that *have* a given *consequence*. To be more precise, the knowledge bases considered in [22] are so-called unfoldable $\mathcal{ALC}$-terminologies, and the unwanted consequences are the unsatisfiability of concepts. The algorithm is an extension of the known tableau-based satisfiability algorithm for $\mathcal{ALC}$ [23], where labels keep track of which axioms are responsible for an assertion to be generated during the run of the algorithm. The authors also coin the name "axiom pinpointing" for the task of computing these minimal subsets. Following Reiter's approach for model-based diagnosis [20], Schlobach [21] uses the minimal subsets that have a given consequence together with the computation of Hitting Sets to compute *maximal subsets* of a given knowledge base that *do not have* a given (unwanted) *consequence*.[2] Whereas the minimal subsets that have the consequence help the user to comprehend why a certain consequence holds, the maximal subsets that do not have the consequence suggest how to change the knowledge base in a minimal way to get rid of a certain unwanted consequence.

The problem of computing minimal (maximal) subsets of a DL knowledge base that have (do not have) a given consequence was actually considered earlier in the context of extending DLs by default rules. In [4], Baader and Hollunder solve this problem by introducing a labeled extension of the tableau-based consistency algorithm for $\mathcal{ALC}$-ABoxes [11], which is very similar to the one described later in [22]. The main difference is that the algorithm described in [4] does not directly compute minimal subsets that have a consequence, but rather a monotone Boolean formula, called *clash formula* in [4], whose variables correspond to the axioms of the knowledge bases and whose minimal satisfying (maximal unsatisfying) valuations correspond to the minimal (maximal) subsets that have (do not have) a given consequence.

The approach of Schlobach and Cornet [22] was extended by Parsia et al. [19] to more expressive DLs, and the one of Baader and Hollunder [4] was extended by Meyer et al. [17] to the case of $\mathcal{ALC}$-terminologies with general concept inclusions (GCIs), which are no longer unfoldable. The choice of the DL $\mathcal{ALC}$ in [4] and [22] was meant to be prototypical, i.e., in both cases the authors assumed that their approach could be easily extended to other DLs and tableau-based algorithms for them. However, the algorithms and proofs are given for $\mathcal{ALC}$ only, and it is not clear to which of the known tableau-based algorithms the approaches really generalize. For example, the pinpointing extension described in [17] follows the approach introduced in [4], but since GCIs require the introduction of so-called

[2] Actually, he considers the complements of these sets, which he calls minimal diagnoses.

blocking conditions into the tableau-based algorithm to ensure termination, there are some new problems to be solved.

Thus, one can ask to which DLs and tableau-based algorithms the approaches described in [4, 22] apply basically without significant changes, and with no need for a new proof of correctness. This paper is a first step towards answering this question. We develop a general approach for extending a tableau-based algorithm to a pinpointing algorithm, which is based on the the ideas underlying the pinpointing algorithm described in [4]. To this purpose, we define a general notion of "tableaux algorithm," which captures many of the known tableau-based algorithms for DLs and Modal Logics,[3] but also other kinds of decision procedures, like the polynomial-time subsumption algorithm for the DL $\mathcal{EL}$ [1]. This notion is simpler than the tableau systems introduced in [3] in the context of translating tableaux into tree automata, and it is not restricted to tableau-based algorithms that generate tree-like structures.

Axiom pinpointing has also been considered in other research areas, though usually not under this name. For example, in the SAT community, people have considered the problem of computing maximally satisfiable and minimally unsatisfiable subsets of a set of propositional formulae. The approaches for computing these sets developed there include special purpose algorithms that call a SAT solver as a black box [16, 7], but also algorithms that extend a resolution-based SAT solver directly [8, 26]. To the best of our knowledge, extensions of tableau-based algorithms have not been considered in this context, and there are no general schemes for extending resolution-based solvers.

In the next section, we define the notions of minimal (maximal) sets having (not having) a given consequence in a general setting, and show some interesting connections between these two notions. In Section 3 we introduce our general notion of a tableau, and in Section 4 we show how to obtain pinpointing extension of such tableaux. Because of the space restriction, we cannot give complete proofs of our results. They can be found in [5].

2 Basic Definitions

Before we can define our general notion of a tableau algorithm, we need to define the general form of inputs to which these algorithms are applied, and the decision problems they are supposed to solve.

Definition 1 (Axiomatized input, c-property). *Let $\mathfrak{I}$ be a set, called the set of* inputs*, and $\mathfrak{T}$ be a set, called the set of* axioms*. An* axiomatized input *over these sets is of the form $(\mathcal{I}, \mathcal{T})$ where $\mathcal{I} \in \mathfrak{I}$ and $\mathcal{T} \in \mathscr{P}_{fin}(\mathfrak{T})$ is a finite subset of $\mathfrak{T}$. A* consequence property (c-property) *is a set $\mathcal{P} \subseteq \mathfrak{I} \times \mathscr{P}_{fin}(\mathfrak{T})$ such that $(\mathcal{I}, \mathcal{T}) \in \mathcal{P}$ implies $(\mathcal{I}, \mathcal{T}') \in \mathcal{P}$ for every $\mathcal{T}' \supseteq \mathcal{T}$.*

[3] Note that these algorithms are decision procedures, i.e., always terminate. Currently, our approach does not cover semi-decision procedures like tableaux procedures for first-order logic.

Intuitively, c-properties on axiomatized inputs are supposed to model consequence relations in logic, i.e., the c-property $\mathcal{P}$ holds if the input $\mathcal{I}$ "follows" from the axioms in $\mathcal{T}$. The monotonicity requirement on c-properties corresponds to the fact that we want to restrict the attention to consequence relations induced by monotonic logics. In fact, for non-monotonic logics, looking at minimal sets of axioms that have a given consequence does not make much sense.

To illustrate Definition 1, assume that $\mathfrak{I}$ is a countably infinite set of propositional variables, and that $\mathfrak{T}$ consists of all Horn clauses over these variables, i.e., implications of the form $p_1 \wedge \ldots \wedge p_n \rightarrow q$ for $n \geq 0$ and $p_1, \ldots, p_n, q \in \mathfrak{I}$. Then the following is a c-property according to the above definition: $\mathcal{P} := \{(p, \mathcal{T}) \mid \mathcal{T} \models p\}$, where $\mathcal{T} \models q$ means that all valuations satisfying all implications in $\mathcal{T}$ also satisfy q. As as concrete example, consider $\Gamma := (p, \mathcal{T})$ where $\mathcal{T}$ consists of the following implications:

$$\mathrm{ax}_1\colon\ \rightarrow q, \qquad \mathrm{ax}_2\colon\ \rightarrow s, \qquad \mathrm{ax}_3\colon\ s \rightarrow q, \qquad \mathrm{ax}_4\colon\ q \wedge s \rightarrow p \tag{1}$$

It is easy to see that $\Gamma \in \mathcal{P}$. Note that Definition 1 also captures the following variation of the above example, where $\mathfrak{I}'$ consist of tuples $(p, \mathcal{T}_1) \in \mathcal{I} \times \mathscr{P}_{fin}(\mathfrak{T})$ and the c-property is defined as $\mathcal{P}' := \{((p, \mathcal{T}_1), \mathcal{T}_2) \mid \mathcal{T}_1 \cup \mathcal{T}_2 \models p\}$. For example, if we take the axiomatized input $\Gamma' := ((p, \{\mathrm{ax}_3, \mathrm{ax}_4\}), \{\mathrm{ax}_1, \mathrm{ax}_2\})$, then $\Gamma' \in \mathcal{P}'$.

Definition 2. *Given an axiomatized input $\Gamma = (\mathcal{I}, \mathcal{T})$ and a c-property $\mathcal{P}$, a set of axioms $\mathcal{S} \subseteq \mathcal{T}$ is called a* minimal axiom set (MinA) *for Γ* w.r.t. *$\mathcal{P}$ if $(\mathcal{I}, \mathcal{S}) \in \mathcal{P}$ and $(\mathcal{I}, \mathcal{S}') \notin \mathcal{P}$ for every $\mathcal{S}' \subset \mathcal{S}$. Dually, a set of axioms $\mathcal{S} \subseteq \mathcal{T}$ is called a* maximal non-axiom set (MaNA) *for Γ* w.r.t. *$\mathcal{P}$ if $(\mathcal{I}, \mathcal{S}) \notin \mathcal{P}$ and $(\mathcal{I}, \mathcal{S}') \in \mathcal{P}$ for every $\mathcal{S}' \supset \mathcal{S}$. The set of all MinA (MaNA) for Γ w.r.t. $\mathcal{P}$ will be denoted as $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ ($\mathsf{MAX}_{\mathcal{P}(\Gamma)}$).*

Note that the notions of MinA and MaNA are only interesting in the case where $\Gamma \in \mathcal{P}$. In fact, otherwise the monotonicity property satisfied by $\mathcal{P}$ implies that $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \emptyset$ and $\mathsf{MAX}_{\mathcal{P}(\Gamma)} = \{\mathcal{T}\}$. In the above example, where we have $\Gamma \in \mathcal{P}$, it is easy to see that $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \{\{\mathrm{ax}_1, \mathrm{ax}_2, \mathrm{ax}_4\},\ \{\mathrm{ax}_2, \mathrm{ax}_3, \mathrm{ax}_4\}\}$. In the variant of the example where only subsets of the facts $\{\mathrm{ax}_1, \mathrm{ax}_2\}$ can be taken, we have $\mathsf{MIN}_{\mathcal{P}'(\Gamma')} = \{\{\mathrm{ax}_2\}\}$.

The set $\mathsf{MAX}_{\mathcal{P}(\Gamma)}$ can be obtained from $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ by computing the minimal hitting sets of $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$, and then complementing these sets [22, 16]. A set $\mathcal{S} \subseteq \mathcal{T}$ is a *minimal hitting set* of $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ if it has a nonempty intersection with every element of $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$, and no strict subset of $\mathcal{S}$ has this property. In our example, the minimal hitting sets of $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ are $\{\mathrm{ax}_1, \mathrm{ax}_3\}, \{\mathrm{ax}_2\}, \{\mathrm{ax}_4\}$, and thus $\mathsf{MAX}_{\mathcal{P}(\Gamma)} = \{\{\mathrm{ax}_2, \mathrm{ax}_4\},\ \{\mathrm{ax}_1, \mathrm{ax}_3, \mathrm{ax}_4\},\ \{\mathrm{ax}_1, \mathrm{ax}_2, \mathrm{ax}_3\}\}$. Intuitively, to get a set of axioms that does not have the consequence, we must remove from $\mathcal{T}$ at least one axiom for every MinA, and thus the minimal hitting sets give us the minimal sets to be removed.

The reduction we have just sketched shows that it is enough to design an algorithm for computing all MinA, since the MaNA can then be obtained by a hitting set computation. It should be noted, however, that this reduction is not polynomial: there may be exponentially many hitting sets of a given collection of sets, and even deciding whether such a collection has a hitting set of cardinality $\leq n$ is an NP-complete problem [9]. Also note that there is a similar reduction involving hitting sets for computing the MinA from all MaNA.

Instead of computing MinA or MaNA, one can also compute the pinpointing formula.[4] To define the pinpointing formula, we assume that every axiom $t \in \mathcal{T}$ is labeled with a unique propositional variable, $\mathsf{lab}(t)$. Let $\mathsf{lab}(\mathcal{T})$ be the set of all propositional variables labeling an axiom in $\mathcal{T}$. A *monotone Boolean formula* over $\mathsf{lab}(\mathcal{T})$ is a Boolean formula using (some of) the variables in $\mathsf{lab}(\mathcal{T})$ and only the connectives conjunction and disjunction. As usual, we identify a propositional *valuation* with the set of propositional variables it makes true. For a valuation $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$, let $\mathcal{T}_\mathcal{V} := \{t \in \mathcal{T} \mid \mathsf{lab}(t) \in \mathcal{V}\}$.

Definition 3 (pinpointing formula). *Given a c-property $\mathcal{P}$ and an axiomatized input $\Gamma = (\mathcal{I}, \mathcal{T})$, a monotone Boolean formula ϕ over $\mathsf{lab}(\mathcal{T})$ is called a* pinpointing formula *for $\mathcal{P}$ and Γ if the following holds for every valuation $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$: $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \in \mathcal{P}$ iff $\mathcal{V}$ satisfies ϕ.*

In our example, we can take $\mathsf{lab}(\mathcal{T}) = \{\mathrm{ax}_1, \ldots, \mathrm{ax}_4\}$ as set of propositional variables. It is easy to see that $(\mathrm{ax}_1 \vee \mathrm{ax}_3) \wedge \mathrm{ax}_2 \wedge \mathrm{ax}_4$ is a pinpointing formula for $\mathcal{P}$ and Γ.

Valuations can be ordered by set inclusion. The following is an immediate consequence of the definition of a pinpointing formula [4].

Lemma 1. *Let $\mathcal{P}$ be a c-property, $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatized input, and ϕ a pinpointing formula for $\mathcal{P}$ and Γ. Then*

$$\begin{aligned} \mathsf{MIN}_{\mathcal{P}(\Gamma)} &= \{\mathcal{T}_\mathcal{V} \mid \mathcal{V} \textit{ is a minimal valuation satisfying } \phi\} \\ \mathsf{MAX}_{\mathcal{P}(\Gamma)} &= \{\mathcal{T}_\mathcal{V} \mid \mathcal{V} \textit{ is a maximal valuation falsifying } \phi\} \end{aligned}$$

This shows that it is enough to design an algorithm for computing a pinpointing formula to obtain all MinA and MaNA. However, like the previous reduction from computing MaNA from MinA, the reduction suggested by the lemma is not polynomial. For example, to obtain $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ from ϕ, one can bring ϕ into disjunctive normal form and then remove disjuncts implying other disjuncts. It is well-known that this can cause an exponential blowup. Conversely, however, the set $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ can directly be translated into the pinpointing formula

$$\bigvee_{\mathcal{S} \in \mathsf{MIN}_{\mathcal{P}(\Gamma)}} \bigwedge_{s \in \mathcal{S}} \mathsf{lab}(s).$$

In our example, the pinpointing formula obtained from the set $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \{\{\mathrm{ax}_1, \mathrm{ax}_2, \mathrm{ax}_4\}, \{\mathrm{ax}_2, \mathrm{ax}_3, \mathrm{ax}_4\}\}$ is $(\mathrm{ax}_1 \wedge \mathrm{ax}_2 \wedge \mathrm{ax}_4) \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3 \wedge \mathrm{ax}_4)$.

[4] This corresponds to the clash formula introduced in [4]. Here, we distinguish between the pinpointing formula, which can be defined independently of a tableau algorithm, and the clash formula, which is induced by a run of a tableau algorithm.

3 A General Notion of Tableaux

Before introducing our general notion of a tableau-based decision procedure, we want to motivate it by first modelling a simple decision procedure for the property $\mathcal{P}$ introduced in the Horn clause example from the previous section, and then sketching extensions to the model that are needed to treat more complex tableau-based decision procedures.

Motivating Examples

To decide whether $(p, \mathcal{T}) \in \mathcal{P}$, we start with the set $A := \{\neg p\}$, and then use the rule

$$\text{If } \{p_1, \ldots, p_n\} \subseteq A \text{ and } p_1 \wedge \ldots \wedge p_n \rightarrow q \in \mathcal{T} \text{ then } A := A \cup \{q\} \tag{2}$$

to extend A until it is saturated, i.e., it can no longer be extended with the above rule. It is easy to see that $(p, \mathcal{T}) \in \mathcal{P}$ (i.e., $\mathcal{T} \models p$) iff this saturated set contains both p and $\neg p$. For example, for the axioms in (1), one can first add s using ax_2, then q using ax_3, and finally p using ax_4. This yields the saturated set $\{\neg p, p, q, s\}$.

Abstracting from particularities, we can say that we have an algorithm that works on a set of *assertions* (in the example, assertions are propositional variables and their negation), and uses rules to extend this set. A *rule* is of the form $(B_0, \mathcal{S}) \rightarrow B_1$ where B_0, B_1 are finite sets of assertions, and $\mathcal{S}$ is a finite set of *axioms* (in the example, axioms are Horn clauses). Given a set of axioms $\mathcal{T}$ and a set of assertions A, this rule is *applicable* if $B_0 \subseteq A$, $\mathcal{S} \subseteq \mathcal{T}$, and $B_1 \not\subseteq A$. Its *application* then extends A to $A \cup B_1$.[5] Our simple Horn clause algorithm always *terminates* in the sense that any sequence of rule applications is finite (since only right-hand sides of implications in $\mathcal{T}$ can be added). After termination, we have a *saturated* set of assertions, i.e., one to which no rule applies. The algorithm *accepts* the input (i.e., says that it belongs to $\mathcal{P}$) iff this saturated set contains a *clash* (in the example, this is the presence of p and $\neg p$ in the saturated set).

The model of a tableau-based decision procedure introduced until now is too simplistic since it does not capture two important phenomena that can be found in tableau algorithms for description and modal logics: non-determinism and assertions with an internal structure. Regarding *non-determinism*, assume that instead of Horn clauses we have more general implications of the form $p_1 \wedge \ldots \wedge p_n \rightarrow q_1 \vee \ldots \vee q_m$ in $\mathcal{T}$. Then, if $\{p_1, \ldots, p_n\} \subseteq A$, we need to choose (don't know non-deterministically) with which of the propositional variables q_j to extend A. In our formal model, the right-hand side of a non-deterministic rule consists of a finite set of sets of assertions rather than a single set of assertions, i.e., non-deterministic rules are of the more general form $(B_0, \mathcal{S}) \rightarrow \{B_1, \ldots, B_m\}$ where $B_0, B_1, \ldots, B_m$ are finite sets of assertions and $\mathcal{S}$ is a finite set of axioms. Instead of working on a single set of assertions, the non-deterministic algorithm

[5] The applicability condition $B_1 \not\subseteq A$ ensures that rule application really *extends* the given set of assertions.

thus works on a finite set $\mathcal{M}$ of sets of assertions. The non-deterministic rule $(B_0, \mathcal{S}) \rightarrow \{B_1, \ldots, B_m\}$ is applicable to $A \in \mathcal{M}$ if $B_0 \subseteq A$ and $\mathcal{S} \subseteq \mathcal{T}$, and its application replaces $A \in \mathcal{M}$ by the finitely many sets $A \cup B_1, \ldots, A \cup B_m$ provided that each of these sets really extends A. For example, if we replace ax_1 and ax_2 in (1) by ax_5: $\rightarrow p \vee s$, then starting with $\{\{\neg p\}\}$, we first get $\{\{\neg p, p\}, \{\neg p, s\}\}$ using ax_5, then $\{\{\neg p, p\}, \{\neg p, s, q\}\}$ using ax_3, and finally $\{\{\neg p, p\}, \{\neg p, s, q, p\}\}$ using ax_4. Since each of these sets contains a clash, the input is accepted.

Regarding the *structure of assertions*, in general it is not enough to use propositional variables. Tableau-based decision procedures in description and modal logic try to build finite models, and thus assertions must be able to describe the relational structure of such models. For example, assertions in tableau algorithms for description logics [6] are of the form $r(a, b)$ and $C(a)$, where r is a role name, C is a concept description, and a, b are individual names. Again abstracting from particularities, a *structured assertion* is thus of the form $P(a_1, \ldots, a_k)$ where P is a k-ary predicate and $a_1, \ldots, a_k$ are constants. As an example of the kind of rules employed by tableau-based algorithms for description logics, consider the rule treating existential restrictions:

$$\text{If } \{(\exists r.C)(x)\} \subseteq A \text{ then } A := A \cup \{r(x, y), C(y)\}. \tag{3}$$

The variables x, y in this rule are place-holders for constants, i.e., to apply the rule to a set of assertions, we must first replace the variables by appropriate constants. Note that y occurs only on the right-hand side of the rule. We will call such a variable a *fresh variable*. Fresh variables must be replaced by *new constants*, i.e., a constant not occurring in the current set of assertions. For example, let $A := \{(\exists r.C)(a), r(a, b)\}$. If we apply the substitution $\sigma := \{x \mapsto a, y \mapsto c\}$ that replaces x by a and y by the new constant c, then the above rule is applicable with σ since $(\exists r.C)(a) \in A$. Its application yields the set of assertions $A' = A \cup \{r(a, c), C(c)\}$. Of course, we do not want the rule to be still applicable to A'. However, to prevent this it is not enough to require that the right-hand side (after applying the substitution) is not contained in the current set of assertions. In fact, this would not prevent us from applying the rule to A' with another new constant, say c'. For this reason, the applicability condition for rules needs to check whether the assertions obtained from the right-hand side by replacing the fresh variables by existing constants yields assertions that are already contained in the current set of assertions.

The Formal Definition

In the following, $\mathcal{V}$ denotes a countably infinite set of *variables*, and $\mathcal{D}$ a countably infinite set of *constants*. A *signature* Σ is a set of predicate symbols, where each predicate $P \in \Sigma$ is equipped with an arity. A *Σ-assertion* is of the form $P(a_1, \ldots, a_n)$ where $P \in \Sigma$ is an n-ary predicate and $a_1, \ldots, a_n \in \mathcal{D}$. Likewise, a *$\Sigma$-pattern* is of the form $P(x_1, \ldots, x_n)$ where $P \in \Sigma$ is an n-ary predicate and $x_1, \ldots, x_n \in \mathcal{V}$. If the signature is clear from the context, we will often just say pattern (assertion). For a set of assertions A (patterns B), $\mathsf{cons}(A)$ ($\mathsf{var}(B)$) denotes the set of constants (variables) occurring in A (B).

A *substitution* is a mapping $\sigma : V \to \mathcal{D}$, where V is a finite set of variables. If B is a set of patterns such that $\mathsf{var}(B) \subseteq V$, then $B\sigma$ denotes the set of assertions obtained from B by replacing each variable by its σ-image. We say that $\sigma : V \to \mathcal{D}$ is a *substitution on* V. The substitution θ on V' *extends* σ on V if $V \subseteq V'$ and $\theta(x) = \sigma(x)$ for all $x \in V$.

Definition 4 (Tableau). *Let $\mathfrak{I}$ be a set of inputs and $\mathfrak{T}$ a set of axioms. A* tableau *for $\mathfrak{I}$ and $\mathfrak{T}$ is a tuple $S = (\Sigma, \cdot^S, \mathcal{R}, \mathcal{C})$ where*

- *Σ is a signature;*
- *$\cdot^S$ is a function that maps every $\mathcal{I} \in \mathfrak{I}$ to a finite set of finite sets of Σ-assertions;*
- *$\mathcal{R}$ is a set of rules of the form $(B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$ where $B_0, \ldots, B_m$ are finite sets of Σ-patterns and $\mathcal{S}$ is a finite set of axioms;*
- *$\mathcal{C}$ is a set of finite sets of Σ-patterns, called clashes.*

Given a rule $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$, the variable y is a *fresh variable* in R if it occurs in one of the sets $B_1, \ldots, B_m$, but not in B_0.

An *S-state* is a pair $\mathfrak{S} = (A, \mathcal{T})$ where A is a finite set of assertions and $\mathcal{T}$ a finite set of axioms. We extend the function $\cdot^S$ to axiomatized inputs by defining $(\mathcal{I}, \mathcal{T})^S := \{(A, \mathcal{T}) \mid A \in \mathcal{I}^S\}$.

Intuitively, on input $(\mathcal{I}, \mathcal{T})$, we start with the initial set $\mathcal{M} = (\mathcal{I}, \mathcal{T})^S$ of S-states, and then use the rules in $\mathcal{R}$ to modify this set. Each rule application picks an S-state $\mathfrak{S}$ from $\mathcal{M}$ and replaces it by finitely many new S-states $\mathfrak{S}_1, \ldots, \mathfrak{S}_m$ that extend the first component of $\mathfrak{S}$. If $\mathcal{M}$ is saturated, i.e., no more rules are applicable to $\mathcal{M}$, then we check whether all the elements of $\mathcal{M}$ contain a clash. If yes, then the input is accepted; otherwise, it is rejected.

Definition 5 (rule application, saturated, clash). *Given an S-state $\mathfrak{S} = (A, \mathcal{T})$, a rule $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$, and a substitution ρ on $\mathsf{var}(B_0)$, this rule is* applicable to $\mathfrak{S}$ with ρ *if (i) $\mathcal{S} \subseteq \mathcal{T}$, (ii) $B_0\rho \subseteq A$, and (iii) for every $i, 1 \leq i \leq m$, and every substitution ρ' on $\mathsf{var}(B_0 \cup B_i)$ extending ρ we have $B_i\rho' \not\subseteq A$.*

Given a set of S-states $\mathcal{M}$ and an S-state $\mathfrak{S} = (A, \mathcal{T}) \in \mathcal{M}$ to which the rule R is applicable with substitution ρ, the application of R to $\mathfrak{S}$ with ρ in $\mathcal{M}$ *yields the new set $\mathcal{M}' = (\mathcal{M} \setminus \{\mathfrak{S}\}) \cup \{(A \cup B_i\sigma, \mathcal{T}) \mid i = 1, \ldots, m\}$, where σ is a substitution on the variables occurring in R that extends ρ and maps the fresh variables of R to distinct new constants, i.e., constants not occurring in A.*

If $\mathcal{M}'$ is obtained from $\mathcal{M}$ by the application of R, then we write $\mathcal{M} \to_{\mathsf{R}} \mathcal{M}'$, or simply $\mathcal{M} \to_S \mathcal{M}'$ if it is not relevant which of the rules of the tableau S was applied. As usual, the reflexive-transitive closure *of $\to_S$ is denoted by $\xrightarrow{*}_S$. A set of S-states $\mathcal{M}$ is called* saturated *if there is no $\mathcal{M}'$ such that $\mathcal{M} \to_S \mathcal{M}'$.*

The S-state $\mathfrak{S} = (A, \mathcal{T})$ contains a clash *if there is a $C \in \mathcal{C}$ and a substitution ρ on $\mathsf{var}(C)$ such that $C\rho \subseteq A$, and the set of S-states $\mathcal{M}$ is* full of clashes *if* all *its elements contain a clash.*

We can now define under what conditions a tableau S is correct for a c-property.

Definition 6 (correctness). *Let $\mathcal{P}$ be a c-property on axiomatized inputs over $\mathfrak{I}$ and $\mathfrak{T}$, and S a tableau for $\mathfrak{I}$ and $\mathfrak{T}$. Then S is* correct for $\mathcal{P}$ *if the following holds for every axiomatized input $\Gamma = (\mathcal{I}, \mathcal{T})$ over $\mathfrak{I}$ and $\mathfrak{T}$:*

1. *S terminates on Γ, i.e., there is no infinite chain of rule applications $\mathcal{M}_0 \rightarrow_S \mathcal{M}_1 \rightarrow_S \mathcal{M}_2 \rightarrow_S \ldots$ starting with $\mathcal{M}_0 := \Gamma^S$.*
2. *For every chain of rule applications $\mathcal{M}_0 \rightarrow_S \ldots \rightarrow_S \mathcal{M}_n$ such that $\mathcal{M}_0 = \Gamma^S$ and $\mathcal{M}_n$ is saturated we have $\Gamma \in \mathcal{P}$ iff $\mathcal{M}_n$ is full of clashes.*

The simple decision procedure sketched in our Horn clause example is a correct tableau in the sense of this definition. More precisely, it is a tableau with unstructured assertions (i.e., the signature contains only nullary predicate symbols) and deterministic rules. It is easy to see that also the polynomial-time subsumption algorithm for the DL $\mathcal{EL}$ and its extensions introduced in [1] can be viewed as a correct deterministic tableau with unstructured assertions. The standard tableau-based decision procedure for concept unsatisfiability in the DL $\mathcal{ALC}$ [23] is a correct tableau that uses structured assertions and has a non-deterministic rule.

In 2. of Definition 6, we require that the algorithm gives the same answer independent of what terminating chain of rule applications is considered. Thus, the choice of which rule to apply next is don't care non-deterministic in a correct tableau. This is important since a need for backtracking over these choices would render a tableau algorithm completely impractical. However, in our framework this is not really an extra requirement on *correct* tableaux: it is built into our definition of rules and clashes.

Proposition 1. *Let Γ be an axiomatized input and $\mathcal{M}_0 := \Gamma^S$. If $\mathcal{M}$ and $\mathcal{M}'$ are saturated sets of S-states such that $\mathcal{M}_0 \xrightarrow{*}_S \mathcal{M}$ and $\mathcal{M}_0 \xrightarrow{*}_S \mathcal{M}'$, then $\mathcal{M}$ is full of clashes iff $\mathcal{M}'$ is full of clashes.*

4 Pinpointing Extensions of General Tableaux

Given a correct tableau, we show how it can be extended to an algorithm that computes a pinpointing formula. As shown in Section 2, all minimal axiom sets (maximal non-axiom sets) can be derived from the pinpointing formula ϕ by computing all minimal (maximal) valuations satisfying (falsifying) ϕ. Recall that, in the definition of the pinpointing formula, we assume that every axiom $t \in \mathcal{T}$ is labeled with a unique propositional variable, $\mathsf{lab}(t)$. The set of all propositional variables labeling an axiom in $\mathcal{T}$ is denoted by $\mathsf{lab}(\mathcal{T})$. In the following, we assume that the symbol $\top$, which always evaluates to true, also belongs to $\mathsf{lab}(\mathcal{T})$. The pinpointing formula is a monotone Boolean formula over $\mathsf{lab}(\mathcal{T})$, i.e., a Boolean formula built from $\mathsf{lab}(\mathcal{T})$ using conjunction and disjunction only.

To motivate our pinpointing extension of general tableaux, we first describe such an extension of the simple decision procedure sketched for our Horn clause example. The main idea is that assertions are also labeled with monotone Boolean formulae. In the example, where $\mathcal{T}$ consists of the axioms of (1) and the axiomatized input is $(p, \mathcal{T})$, the initial set of assertions consists of $\neg p$. The label of this initial assertion is $\top$ since its presence depends only on the input p, and not on any of the axioms. By applying the rule (2) using axiom ax_2, we can add the assertion s. Since the addition of this assertion depends on the presence of ax_2, it receives label ax_2. Then we can use ax_3 to add q. Since this addition depends on the presence of ax_3 and of the assertion s, which has label ax_2, the label of this new assertion is $\mathrm{ax}_2 \wedge \mathrm{ax}_3$. There is, however, also another possibility to generate the assertion q: apply the rule (2) using axiom ax_1. In a "normal" run of the tableau algorithm, the rule would not be applicable since it would add an assertion that is already there. However, in the pinpointing extension we need to register this alternative way of generating q. Therefore, the rule is applicable using ax_1, and its application changes the label of the assertion q from $\mathrm{ax}_2 \wedge \mathrm{ax}_3$ to $\mathrm{ax}_1 \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3)$. Finally, we can use ax_4 to add the assertion p. The label of this assertion is $\mathrm{ax}_4 \wedge \mathrm{ax}_2 \wedge (\mathrm{ax}_1 \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3))$ since the application of the rule depends on the presence of ax_4 as well as the assertions s and q. The presence of both p and $\neg p$ gives us a clash, which receives label $\top \wedge \mathrm{ax}_4 \wedge \mathrm{ax}_2 \wedge (\mathrm{ax}_1 \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3))$. This so-called clash formula is the output of the extended algorithm. Obviously, it is equivalent to the pinpointing formula $(\mathrm{ax}_1 \vee \mathrm{ax}_3) \wedge \mathrm{ax}_2 \wedge \mathrm{ax}_4$ that we have constructed by hand in Section 2.

The Formal Definition

Given a tableau $S = (\Sigma, \cdot^S, \mathcal{R}, \mathcal{C})$ that is correct for the c-property $\mathcal{P}$, we show how the algorithm for deciding $\mathcal{P}$ induced by S can be modified to an algorithm that computes a pinpointing formula for $\mathcal{P}$. Given an axiomatized input $\Gamma = (\mathcal{I}, \mathcal{T})$, the modified algorithm also works on sets of S-states, but now every assertion a occurring in the assertion component of an S-state is equipped with a label $\mathsf{lab}(a)$, which is a monotone Boolean formula over $\mathsf{lab}(\mathcal{T})$. We call such S-states *labeled S-states.* In the initial set of S-states $\mathcal{M} = (\mathcal{I}, \mathcal{T})^S$, every assertion is labeled with $\top$.

The definition of rule application must take the labels of assertions and axioms into account. Let A be a set of labeled assertions and ψ a monotone Boolean formula. We say that the assertion a is *ψ-insertable into A* if (i) either $a \notin A$, or (ii) $a \in A$, but $\psi \not\models \mathsf{lab}(a)$. Given a set B of assertions and a set A of labeled assertions, the set of *ψ-insertable elements of B into A* is defined as $\mathsf{ins}_\psi(B, A) := \{b \in B \mid b \text{ is } \psi\text{-insertable into } A\}$. By ψ-inserting these insertable elements into A, we obtain the following new set of labeled assertions: $A \Cup_\psi B := A \cup \mathsf{ins}_\psi(B, A)$, where each assertion $a \in A \setminus \mathsf{ins}_\psi(B, A)$ keeps its old label $\mathsf{lab}(a)$, each assertion in $\mathsf{ins}_\psi(B, A) \setminus A$ gets label ψ, and each assertion $b \in A \cap \mathsf{ins}_\psi(B, A)$ gets the new label $\psi \vee \mathsf{lab}(b)$.

Definition 7 (pinpointing rule application). *Given a labeled S-state $\mathfrak{S} = (A, \mathcal{T})$, a rule $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$, and a substitution ρ on $\mathsf{var}(B_0)$, this rule is* pinpointing applicable to $\mathfrak{S}$ with ρ *if (i) $\mathcal{S} \subseteq \mathcal{T}$, (ii) $B_0\rho \subseteq A$, and (iii) for every $i, 1 \leq i \leq m$, and every substitution ρ' on $\mathsf{var}(B_0 \cup B_i)$ extending ρ we have $\mathsf{ins}_\psi(B_i\rho', A) \neq \emptyset$, where $\psi := \bigwedge_{b \in B_0} \mathsf{lab}(b\rho) \wedge \bigwedge_{s \in \mathcal{S}} \mathsf{lab}(s)$.*

Given a set of labeled S-states $\mathcal{M}$ and a labeled S-state $\mathfrak{S} \in \mathcal{M}$ to which the rule R is pinpointing applicable with substitution ρ, the pinpointing application of R to $\mathfrak{S}$ with ρ in $\mathcal{M}$ *yields the new set* $\mathcal{M}' = (\mathcal{M} \setminus \{\mathfrak{S}\}) \cup \{(A \Cup_\psi B_i\sigma, \mathcal{T}) \mid i = 1, \ldots, m\}$*, where the formula* ψ *is defined as above and* σ *is a substitution on the variables occurring in* R *that extends* ρ *and maps the fresh variables of* R *to distinct new constants.*

If $\mathcal{M}'$ *is obtained from* $\mathcal{M}$ *by the pinpointing application of* R*, then we write* $\mathcal{M} \rightarrow_{\mathsf{R}^{pin}} \mathcal{M}'$*, or simply* $\mathcal{M} \rightarrow_{S^{pin}} \mathcal{M}'$ *if it is not relevant which of the rules of the tableau* S *was applied. As before, the* reflexive-transitive closure *of* $\rightarrow_{S^{pin}}$ *is denoted by* $\xrightarrow{*}_{S^{pin}}$*. A set of labeled* S*-states* $\mathcal{M}$ *is called* pinpointing saturated *if there is no* $\mathcal{M}'$ *such that* $\mathcal{M} \rightarrow_{S^{pin}} \mathcal{M}'$*.*

To illustrate the definition of rule application, let us look back at the example from the beginning of this section. There, we have looked at a situation where the current set of assertions is $A := \{\neg p, s, q\}$ where $\mathsf{lab}(\neg p) = \top$, $\mathsf{lab}(s) = \mathrm{ax}_2$, and $\mathsf{lab}(q) = \mathrm{ax}_2 \wedge \mathrm{ax}_3$. In this situation, the rule (1) is pinpointing applicable using ax_1. In fact, in this case the formula ψ is simply ax_1. Since this formula does not imply $\mathsf{lab}(q) = \mathrm{ax}_2 \wedge \mathrm{ax}_3$, the assertion q is ψ-insertable into A. Its insertion changes the label of q to $\mathrm{ax}_1 \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3)$.

Consider a chain of pinpointing rule applications $\mathcal{M}_0 \rightarrow_{S^{pin}} \ldots \rightarrow_{S^{pin}} \mathcal{M}_n$ such that $\mathcal{M}_0 = \Gamma^S$ for an axiomatized input Γ and $\mathcal{M}_n$ is pinpointing saturated. The label of an assertion in $\mathcal{M}_n$ expresses which axioms are needed to obtain this assertion. A clash in an S-state of $\mathcal{M}_n$ depends on the joint presence of certain assertions. Thus, we define the label of the clash as the conjunction of the labels of these assertions. Since it is enough to have just one clash per S-state $\mathfrak{S}$, the labels of different clashes in $\mathfrak{S}$ are combined disjunctively. Finally, since we need a clash in every S-state of $\mathcal{M}_n$, the formulae obtained from the single S-states are again conjoined.

Definition 8 (clash set, clash formula). *Let* $\mathfrak{S} = (A, \mathcal{T})$ *be a labeled* S*-state and* $A' \subseteq A$*. Then* A' *is a* clash set in $\mathfrak{S}$ *if there is a clash* $C \in \mathcal{C}$ *and a substitution* ρ *on* $\mathsf{var}(C)$ *such that* $A' = C\rho$*. The* label *of this clash set is* $\psi_{A'} := \bigwedge_{a \in A'} \mathsf{lab}(a)$*.*

Let $\mathcal{M} = \{\mathfrak{S}_1, \ldots, \mathfrak{S}_n\}$ *be a set of labeled* S*-states. The* clash formula induced by $\mathcal{M}$ *is defined as*

$$\psi_{\mathcal{M}} := \bigwedge_{i=1}^{n} \bigvee_{A' \text{ clash set in } \mathfrak{S}_i} \psi_{A'}.$$

Recall that, given a set $\mathcal{T}$ of labeled axioms, a propositional valuation $\mathcal{V}$ induces the subset $\mathcal{T}_{\mathcal{V}} := \{t \in \mathcal{T} \mid \mathsf{lab}(t) \in \mathcal{V}\}$ of $\mathcal{T}$. Similarly, for a set A of labeled assertions, the valuation $\mathcal{V}$ induces the subset $A_{\mathcal{V}} := \{a \in A \mid \mathcal{V} \text{ satisfies } \mathsf{lab}(a)\}$. Given a labeled S-state $\mathfrak{S} = (A, \mathcal{T})$ we define its $\mathcal{V}$*-projection* as $\mathcal{V}(\mathfrak{S}) := (A_{\mathcal{V}}, \mathcal{T}_{\mathcal{V}})$. The notion of a projection is extended to sets of S-states $\mathcal{M}$ in the obvious way: $\mathcal{V}(\mathcal{M}) := \{\mathcal{V}(\mathfrak{S}) \mid \mathfrak{S} \in \mathcal{M}\}$. The following lemma is an easy consequence of the definition of the clash formula:

Lemma 2. *Let* $\mathcal{M}$ *be a finite set of labeled* S*-states and* $\mathcal{V}$ *a propositional valuation. Then we have that* $\mathcal{V}$ *satisfies* $\psi_{\mathcal{M}}$ *iff* $\mathcal{V}(\mathcal{M})$ *is full of clashes.*

There is also a close connection between pinpointing saturatedness of a set of labeled S-states and saturatedness of its projection:

Lemma 3. *Let $\mathcal{M}$ be a finite set of labeled S-states and $\mathcal{V}$ a propositional valuation. If $\mathcal{M}$ is pinpointing saturated, then $\mathcal{V}(\mathcal{M})$ is saturated.*

Given a tableau that is correct for a property $\mathcal{P}$, its pinpointing extension is correct in the sense that the clash formula induced by the pinpointing saturated set computed by a terminating chain of pinpointing rule applications is indeed a pinpointing formula for $\mathcal{P}$ and the input.

Theorem 1 (correctness of pinpointing). *Let $\mathcal{P}$ be a c-property on axiomatized inputs over $\mathfrak{I}$ and $\mathfrak{T}$, and S a correct tableau for $\mathcal{P}$. Then the following holds for every axiomatized input $\Gamma = (\mathcal{I}, \mathcal{T})$ over $\mathfrak{I}$ and $\mathfrak{T}$:*

> *For every chain of rule applications $\mathcal{M}_0 \rightarrow_{S^{pin}} \ldots \rightarrow_{S^{pin}} \mathcal{M}_n$ such that $\mathcal{M}_0 = \Gamma^S$ and $\mathcal{M}_n$ is pinpointing saturated, the clash formula $\psi_{\mathcal{M}_n}$ induced by $\mathcal{M}_n$ is a pinpointing formula for $\mathcal{P}$ and Γ.*

To prove this theorem, we want to consider projections of chains of pinpointing rule applications to chains of "normal" rule applications. Unfortunately, things are not as simple as one might hope for since in general $\mathcal{M} \rightarrow_{S^{pin}} \mathcal{M}'$ does not imply $\mathcal{V}(\mathcal{M}) \rightarrow_S \mathcal{V}(\mathcal{M}')$. First, the assertions and axioms to which the pinpointing rule was applied in $\mathcal{M}$ may not be present in the projection $\mathcal{V}(\mathcal{M})$ since $\mathcal{V}$ does not satisfy their labels. Thus, we may also have $\mathcal{V}(\mathcal{M}) = \mathcal{V}(\mathcal{M}')$. Second, a pinpointing application of a rule may change the projection (i.e., $\mathcal{V}(\mathcal{M}) \neq \mathcal{V}(\mathcal{M}')$), although this change does not correspond to a normal application of this rule to $\mathcal{V}(\mathcal{M})$. For example, consider the tableau rule (3) treating existential restrictions in description logics, and assume that we have the assertions $(\exists r.C)(a)$ with label ax_1 and $r(a,b), C(b)$ with label ax_2. Then the rule (3) is pinpointing applicable, and its application adds the new assertions $r(a,c), C(c)$ with label ax_1, where c is a new constant. If $\mathcal{V}$ is a valuation that makes ax_1 and ax_2 true, then the $\mathcal{V}$ projection of our set of assertions contains $(\exists r.C)(a), r(a,b), C(b)$. Thus rule (3) is not applicable, and no new individual c is introduced. To overcome this second problem, we define a modified version of rule application, where the applicability condition (iii) from Definition 5 is removed.

Definition 9 (modified rule application). *Given an S-state $\mathfrak{S} = (A, \mathcal{T})$, a rule $\mathsf{R} : (B_0, \mathcal{S}) \rightarrow \{B_1, \ldots, B_m\}$, and a substitution ρ on $\mathsf{var}(B_0)$, this rule is m-*applicable *to $\mathfrak{S}$* with *ρ if (i) $\mathcal{S} \subseteq \mathcal{T}$ and (ii) $B_0\rho \subseteq A$. In this case, we write $\mathcal{M} \rightarrow_{S^m} \mathcal{M}'$ if $\mathfrak{S} \in \mathcal{M}$ and $\mathcal{M}' = (\mathcal{M} \setminus \{\mathfrak{S}\}) \cup \{(A \cup B_i\sigma, \mathcal{T}) \mid i = 1, \ldots, m\}$, where σ is a substitution on the variables occurring in R that extends ρ and maps the fresh variables of R to distinct new constants.*

The next lemma relates modified rule application with "normal" rule application, on the one hand, and pinpointing rule application on the other hand. Note that "saturated" in the formulation of the first part of the lemma means saturated w.r.t. $\rightarrow_S$, as introduced in Definition 5.

Lemma 4. *Let $\Gamma = (\mathcal{I}, \mathcal{T})$ be an axiomatized input and $\mathcal{M}_0 = \Gamma^S$.*

1. *Assume that $\mathcal{M}_0 \xrightarrow{*}_S \mathcal{M}$ and $\mathcal{M}_0 \xrightarrow{*}_{S^m} \mathcal{M}'$ and that $\mathcal{M}$ and $\mathcal{M}'$ are saturated finite sets of S-states. Then $\mathcal{M}$ is full of clashes iff $\mathcal{M}'$ is full of clashes.*
2. *Assume that $\mathcal{M}$ and $\mathcal{M}'$ are finite sets of labeled S-states, and that $\mathcal{V}$ is a propositional valuation. Then $\mathcal{M} \rightarrow_{S^{pin}} \mathcal{M}'$ implies $\mathcal{V}(\mathcal{M}) \rightarrow_{S^m} \mathcal{V}(\mathcal{M}')$ or $\mathcal{V}(\mathcal{M}) = \mathcal{V}(\mathcal{M}')$. In particular, this shows that $\mathcal{M}_0 \xrightarrow{*}_{S^{pin}} \mathcal{M}$ implies $\mathcal{V}(\mathcal{M}_0) \xrightarrow{*}_{S^m} \mathcal{V}(\mathcal{M})$.*

We are now ready to prove Theorem 1. Let $\Gamma = (\mathcal{I}, \mathcal{T})$ be an axiomatized input, and assume that $\mathcal{M}_0 \rightarrow_{S^{pin}} \ldots \rightarrow_{S^{pin}} \mathcal{M}_n$ such that $\mathcal{M}_0 = \Gamma^S$ and $\mathcal{M}_n$ is pinpointing saturated. We must show that the clash formula $\psi := \psi_{\mathcal{M}_n}$ is a pinpointing formula for the property $\mathcal{P}$. This is an immediate consequence of the next two lemmas.

Lemma 5. *If $(\mathcal{I}, \mathcal{T}_{\mathcal{V}}) \in \mathcal{P}$ then $\mathcal{V}$ satisfies ψ.*

Proof. Let $\mathcal{N}_0 := (\mathcal{I}, \mathcal{T}_{\mathcal{V}})^S$. Since S terminates on every input, there is a saturated set $\mathcal{N}$ such that $\mathcal{N}_0 \xrightarrow{*}_S \mathcal{N}$. Since S is correct for $\mathcal{P}$ and $(\mathcal{I}, \mathcal{T}_{\mathcal{V}}) \in \mathcal{P}$, we know that $\mathcal{N}$ is full of clashes.

By 2. of Lemma 4, $\mathcal{M}_0 \xrightarrow{*}_{S^{pin}} \mathcal{M}_n$ implies $\mathcal{V}(\mathcal{M}_0) \xrightarrow{*}_{S^m} \mathcal{V}(\mathcal{M}_n)$. In addition, we know that $\mathcal{V}(\mathcal{M}_0) = \mathcal{N}_0$, and Lemma 3 implies that $\mathcal{V}(\mathcal{M}_n)$ is saturated. Thus, 1. of Lemma 4, together with the fact that $\mathcal{N}$ is full of clashes, implies that $\mathcal{V}(\mathcal{M}_n)$ is full of clashes.

By Lemma 2, this implies that $\mathcal{V}$ satisfies $\psi = \psi_{\mathcal{M}_n}$. □

Lemma 6. *If $\mathcal{V}$ satisfies ψ then $(\mathcal{I}, \mathcal{T}_{\mathcal{V}}) \in \mathcal{P}$.*

Proof. Consider again a chain of rule applications $\mathcal{N}_0 = (\mathcal{I}, \mathcal{T}_{\mathcal{V}})^S \xrightarrow{*}_S \mathcal{N}$ where $\mathcal{N}$ is saturated. We have $(\mathcal{I}, \mathcal{T}_{\mathcal{V}}) \in \mathcal{P}$ if we can show that $\mathcal{N}$ is full of clashes.

As in the proof of the previous lemma, we have that $\mathcal{V}(\mathcal{M}_0) \xrightarrow{*}_{S^m} \mathcal{V}(\mathcal{M}_n)$, $\mathcal{V}(\mathcal{M}_0) = \mathcal{N}_0$, and $\mathcal{V}(\mathcal{M}_n)$ is saturated. Since $\mathcal{V}$ satisfies ψ, Lemma 2 implies that $\mathcal{V}(\mathcal{M}_n)$ is full of clashes.

By 1. of Lemma 4, this implies that $\mathcal{N}$ is full of clashes. □

This completes the proof of Theorem 1. The theorem considers a terminating chain of pinpointing rule applications. Unfortunately, termination of a tableau S in general does not imply termination of its pinpointing extension. The reason is that a rule may be pinpointing applicable in cases where it is not applicable in the normal sense (see the discussion above Definition 9).

Example 1. Consider the tableau S that has the following three rules

$$\begin{aligned}
&\mathsf{R}_1 : (\{P(x)\}, \{\mathrm{ax}_1\}) \rightarrow \{\{P'(x), Q_1(x)\}\},\\
&\mathsf{R}_2 : (\{P(x)\}, \{\mathrm{ax}_2\}) \rightarrow \{\{P'(x), Q_2(x)\}\},\\
&\mathsf{R}_3 : (\{P'(x)\}, \emptyset) \rightarrow \{\{r(x,y), P'(y)\}, \{Q_1(x)\}, \{Q_2(x)\}\},
\end{aligned}$$

and where the function $\cdot^S$ maps every input $\mathcal{I} \in \mathfrak{I}$ to the singleton set $\{\{P(a)\}\}$, and the set of axioms is $\mathfrak{T} = \{\mathrm{ax}_1, \mathrm{ax}_2\}$.

For any axiomatized input $\Gamma = (\mathcal{I}, \mathcal{T})$, we have $\Gamma^S = \{(\{P(a)\}, \mathcal{T})\}$, and thus R_3 is not applicable to Γ^S. Depending on which axioms are contained in $\mathcal{T}$, the rules R_1 and/or R_2 may be applicable. However, their application introduces $Q_1(a)$ or $Q_2(a)$ into the set of assertions, and thus the non-deterministic and potentially non-terminating rule R_3 is not applicable. Consequently, S terminates on every axiomatized input Γ.

It is possible, however, to construct an infinite chain of pinpointing rule applications starting with $\Gamma^S = \{(\{P(a)\}, \{\mathrm{ax}_1, \mathrm{ax}_2\})\}$ where $\mathsf{lab}(P(a)) = \top$. In fact, we can first apply the rule R_1. This adds the assertions $P'(a)$ and $Q_1(a)$, both with label ax_1. An application of the rule R_2 adds the assertion $Q_2(a)$ with label ax_2, and it modifies the label of the assertion $P'(a)$ to $\mathsf{lab}(P'(a)) = \mathrm{ax}_1 \vee \mathrm{ax}_2$. At this point, we have reached an S-state $\mathfrak{S}$ containing the assertions $P(a), P'(a), Q_1(a), Q_2(a)$ with labels $\mathsf{lab}(P(a)) = \top, \mathsf{lab}(P'(a)) = \mathrm{ax}_1 \vee \mathrm{ax}_2, \mathsf{lab}(Q_1(a)) = \mathrm{ax}_1$, and $\mathsf{lab}(Q_2(a)) = \mathrm{ax}_2$. The rule R_3 is pinpointing applicable to this S-state. Indeed, although both $Q_1(a)$ and $Q_2(a)$ are contained in the assertion set of $\mathfrak{S}$, their labels are not implied by $\mathsf{lab}(P'(a))$. The application of R_3 to $\mathfrak{S}$ replaces $\mathfrak{S}$ by three new S-states. One of these new S-states contains the assertion $P'(b)$ for a new constant b. Thus, R_3 is again applicable to this S-state, generating a new S-state with an assertion $P'(c)$ for a new constant c, etc. It is easy to see that this leads to an infinite chain of pinpointing rule applications.

The example shows that, to ensure termination of the pinpointing extension of a tableau, termination of this tableau on every axiomatized input is not sufficient. From the example one also gets the intuition that the reason why the tableau terminates, but its pinpointing extensions does not, is related to the applicability condition for *non*-deterministic rules. In fact, this condition ensures that the rule R_3, which causes non-termination, cannot be applied. The reason is that the assertion $P'(a)$ can only be generated together with $Q_1(a)$ or $Q_2(a)$. Once $Q_i(a)$ for $i \in \{1, 2\}$ is present, the definition of rule appication prevents R_3 from being applied. In the pinpointing case, this is no longer true since the labels must be taken into account, and thus the pure presence of $Q_i(a)$ is not sufficient to prevent the application of R_3.

Unfortunately, non-deterministic rules are not the only culprit that prevent transfer of termination. The following example introduces a terminating tableau with purely deterministic rules whose pinpointing extension is non-terminating.

Example 2. Consider the tableau S that has the following three rules

$$\begin{aligned}
&\mathsf{R}_1 : (\{P(x)\}, \{\mathrm{ax}_1\}) \to R,\\
&\mathsf{R}_2 : (\{P(x)\}, \{\mathrm{ax}_2\}) \to R,\\
&\mathsf{R}_3 : (\{Q_1(x), Q_2(y)\}, \emptyset) \to \{\{r(x, y, z), Q_1(y), Q_2(z)\}\},
\end{aligned}$$

with $R = \{Q_1(x), Q_1(y), Q_2(x), Q_2(y), r(x, x, x), r(x, y, x), r(y, x, x), r(y, y, x)\}$, and where the function $\cdot^S$ maps every input $\mathcal{I} \in \mathfrak{I}$ to the singleton set $\{\{P(a)\}\}$, and the set of axioms is $\mathfrak{T} = \{\mathrm{ax}_1, \mathrm{ax}_2\}$. For any axiomatized input $\Gamma = (\mathcal{I}, \mathcal{T})$, we have $\Gamma^S = \{(\{P(a)\}, \mathcal{T})\}$. Depending on which axioms are contained in $\mathcal{T}$, the rules R_1 and/or R_2 may be applicable, but R_3 is not. Notice that R_1 and R_2 have the same right-hand side, and thus application of R_1 or R_2 to Γ^S leads to the same S-state, modulo the chosen new constant introduced for the fresh variable y. Suppose we apply one of these two rules, introducing b as the new constant. Then the resulting S-state is be given by $\mathfrak{S} = (\mathcal{A}, \mathcal{T})$ where

$$\mathcal{A} = \{P(a), Q_1(a), Q_1(b), Q_2(a), Q_2(b), r(a,a,a), r(a,b,a), r(b,a,a), r(b,b,a)\}.$$

No rule is applicable to $\mathfrak{S}$. In fact, in order to apply rule R_1 or R_2, the only way to satisfy Condition (ii) in the definition of rule application is to use a valuation that maps x to the constant a. Extending this valuation to map y to a as well violates Condition (iii) of the definition of rule application since the assertions $Q_1(a), Q_2(a)$ and $r(a,a,a)$ were already introduced by the first rule application. To satisfy Condition (ii) for rule R_3, we must choose a valuation ρ mapping x to a or b and y to a or b. In any case, the assertions $r(\rho(x), \rho(y), a), Q_1(\rho(x))$ and $Q_2(a)$ belong to $\mathcal{A}$, and thus extending ρ by mapping z to a violates Condition (iii). This shows that S indeed terminates on every axiomatized input.

It is possible to construct an infinite chain of pinpointing rule applications starting with $\Gamma^S = \{(\{P(a)\}, \{\mathrm{ax}_1, \mathrm{ax}_2\})\}$ where $\mathsf{lab}(P(a)) = \top$. We can first apply rule R_1 leading to the S-state $\mathfrak{S}$ described above, where all the assertions, except $P(a)$ are labeled with ax_1. Rule R_2 is pinpointing applicable to $\mathfrak{S}$ since, although there is an extension of the valuation such that all the assertions exist already in $\mathfrak{S}$, these assertions are labeled with the formula ax_1, which is not implied by ax_2. The pinpointing application of R_2 to $\mathfrak{S}$ adds the assertions $Q_1(c), Q_2(c), r(a,c,a), r(c,a,a), r(c,c,a)$ with label ax_2, and modifies the label of $Q_1(a), Q_2(a), r(a,a,a)$ to $\mathrm{ax}_1 \vee \mathrm{ax}_2$. We can now apply R_3 to the resulting S-state $\mathfrak{S}'$ with the valuation ρ mapping x and y to b and c, respectively. Since the S-state $\mathfrak{S}'$ does not contain any assertion of the form $r(b, c, \_)$, Condition (iii) is not violated anymore. This rule application adds the assertions $r(b,c,d), Q_2(d)$ with label $\mathrm{ax}_1 \wedge \mathrm{ax}_2$. It is easy to see that the rule R_3 can now be repeatedly applied, producing an infinite chain of pinpointing rule applications.

5 Conclusion

We have introduced a general notion of tableaux, and have shown that tableaux that are correct for a consequence property can be extended such that a terminating run of the extended procedure computes a pinpointing formula. This formula can then be used to derive minimal axiom sets and maximal non-axiom sets from it.

We have also shown that, in general, termination of a tableau does not imply termination of its pinpointing extension, even if all tableau rules are deterministic. The most important topic for future research is to address the termination issue: under what additional conditions does termination of a tableau transfer to its pinpointing extension?

In addition, our current framework has two restrictions that we will try to overcome in future work. First, our tableau rules always extend the current set of assertions. We do not allow for rules that can modify existing assertions. Thus, tableau-based algorithms that identify constants, like the rule treating at-most number restrictions in description logics (see, e.g., [6]), cannot be modelled. A similar problem occurs for the tableau systems introduced in [3]. There, it was solved by modifying the definition of rule application by allowing rules that introduce new individuals (in our notation: rules with fresh variables) to reuse existing individuals. However, this makes such rules intrinsically non-deterministic. In our setting, we believe that we can solve this problem more elegantly by introducing equality and inequality predicates.

Second, our approach currently assumes that a correct tableau always terminates, without considering additional blocking conditions. As shown in [17], extending a tableau with blocking to a pinpointing algorithm requires some additional effort. Solving this for the case of general tableaux will be a second important direction for future research.

References

[1] Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of IJCAI 2005, pp. 364–369. Morgan Kaufmann, Los Altos (2005)

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

[3] Baader, F., Hladik, J., Lutz, C., Wolter, F.: From tableaux to automata for description logics. Fundamenta Informaticae 57(2–4), 247–279 (2003)

[4] Baader, F., Hollunder, B.: Embedding defaults into terminological knowledge representation formalisms. J. of Automated Reasoning 14, 149–180 (1995)

[5] Baader, F., Penaloza, R.: Axiom pinpointing in general tableaux. LTCS-Report 07-01, TU Dresden, Germany, (2007) See `http://lat.inf.tu-dresden.de/research/reports.html`

[6] Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Studia Logica 69, 5–40 (2001)

[7] Bailey, J., Stuckey, P.J.: Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In: Hermenegildo, M.V., Cabeza, D. (eds.) Practical Aspects of Declarative Languages. LNCS, vol. 3350, pp. 174–186. Springer, Heidelberg (2005)

[8] Davydov, G., Davydova, I., Büning, H.K.: An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. Ann. of Mathematics and Artificial Intelligence 23(3–4), 229–245 (1998)

[9] Garey, M.R., Johnson, D.S.: Computers and Intractability — A guide to NP-completeness. W. H. Freeman and Company, San Francisco (CA, USA) (1979)

[10] Haarslev, V., Möller, R.: RACER system description. In: Proc. of IJCAR 2001 (2001)

[11] Hollunder, B.: Hybrid inferences in KL-ONE-based knowledge representation systems. In: Proc. of German Workshop on AI, pp. 38–47. Springer, Heidelberg (1990)

[12] Horrocks, I.: Using an expressive description logic: FaCT or fiction. In: Proc. of KR'98, pp. 636–647 (1998)

[13] Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. J. of Web Semantics 1(1), 7–26 (2003)

[14] Kalyanpur, A., Parsia, B., Sirin, E., Cuenca-Grau, B., Hendler, J.: Swoop: A Web ontology editing browser. J. of Web Semantics, 4(2) (2005)

[15] Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.A.: The Protégé OWL plugin: An open development environment for semantic web applications. In: Proceedings of the Third Int. Semantic Web Conf. Hiroshima, Japan (2004)

[16] Liffiton, M.H., Sakallah, K.A.: On finding all minimally unsatisfiable subformulas. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 173–186. Springer, Heidelberg (2005)

[17] Meyer, T., Lee, K., Booth, R., Pan, J.Z.: Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In: Proc. of AAAI 2006. AAAI Press/The MIT Press (2006)

[18] Oberle, D., Volz, R., Motik, B., Staab, S.: An extensible ontology software environment. In: Handbook on Ontologies, International Handbooks on Information Systems, pp. 311–333. Springer, Heidelberg (2004)

[19] Parsia, B., Sirin, E., Kalyanpur, A.: Debugging OWL ontologies. In: Proc. of WWW'05, pp. 633–640. ACM, New York (2005)

[20] Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)

[21] Schlobach, S.: Diagnosing terminologies. In: Proc. of AAAI 2005, AAAI Press/The MIT Press, pp. 670–675 (2005)

[22] Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Proc. of IJCAI 2003, Acapulco, Mexico, pp. 355–362. Morgan Kaufmann, Los Altos (2003)

[23] Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artificial Intelligence 48(1), 1–26 (1991)

[24] Sirin, E., Parsia, B.: Pellet: An OWL DL reasoner. In: Proc. of DL 2004, pp. 212–213 (2004)

[25] Spackman, K.A., Campbell, K.E., Cote, R.A.: SNOMED RT: A reference terminology for health care. J. of the American Medical Informatics Association, pp. 640–644, Fall Symposium Supplement (1997)

[26] Zhang, L., Malik, S.: Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In: Proc. of DATE'03, pp. 10880–10885. IEEE Computer Society Press, Washington (2003)

Proof Theory for First Order Łukasiewicz Logic

Matthias Baaz[1] and George Metcalfe[2]

[1] Institute of Discrete Mathematics and Geometry, Technical University Vienna,
Wiedner Hauptstrasse 8-10, A-1040 Wien, Austria
baaz@logic.at

[2] Department of Mathematics, Vanderbilt University
1326 Stevenson Center, Nashville TN 37240, USA
george.metcalfe@vanderbilt.edu

Abstract. An approximate Herbrand theorem is proved and used to establish Skolemization for first-order Łukasiewicz logic. Proof systems are then defined in the framework of hypersequents. In particular, extending a hypersequent calculus for propositional Łukasiewicz logic with usual Gentzen quantifier rules gives a calculus that is complete with respect to interpretations in safe MV-algebras, but lacks cut-elimination. Adding an infinitary rule to the cut-free version of this calculus gives a system that is complete for the full logic. Finally, a cut-free calculus with finitary rules is obtained for the one-variable fragment by relaxing the eigenvariable condition for quantifier rules.

1 Introduction

The infinite-valued Łukasiewicz logic **Ł**, introduced for philosophical reasons by Jan Łukasiewicz in [12], is among the most important and widely studied of all non-classical logics. Along with Gödel logic and Product logic, it is viewed as one of the fundamental "t-norm based" fuzzy logics (see [8] for details). It is also the logic characterized by the class of MV-algebras (for an in-depth treatment see the book [7]) and, via McNaughton's theorem [14], "piecewise continuous" linear functions on $[0, 1]$. For a good historical overview of the main developments and results for Łukasiewicz and related many-valued logics consult [13].

Given the pre-eminence of **Ł** in Fuzzy Logic and other non-classical contexts, it is perhaps disappointing that there is relatively little understanding of its first-order counterpart $\forall$**Ł**. This is partly due to the fact that the valid formulas of $\forall$**Ł** are not recursively enumerable, a result first obtained by Scarpellini [22] and later sharpened by Ragaz to Π_2-completeness [20]. On the other hand, axiomatizations of $\forall$**Ł** with infinitary rules have been given by Hay [10], Belluce and Chang [5,4] (see also Mostowski [16]), and Hájek [8]. Various fragments of $\forall$**Ł** have also been investigated. Validity and satisfiability in the one-variable fragment was proved to be decidable by Rutledge [21], while satisfiability for the monadic fragment was shown to be Π_1-complete by Ragaz [20]; the complexity of validity for this fragment being an open problem. Validity for the fragment given by safe MV-algebras is Σ_1-complete [5]. Also of interest is a decidable "fuzzy description logic" fragment of $\forall$**Ł** investigated by Hájek in [9].

The aim of this paper is to make a preliminary proof-theoretic investigation of first-order Łukasiewicz logic $\forall$**Ł**, the grander purpose being to understand something more

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 28–42, 2007.

about this logic by considering it from an algorithmic perspective. To this end, we begin by giving a simple proof (a more complicated proof is given by Novak in [19]) of the fact that while the usual Herbrand theorem does not hold for $\forall$**Ł**, an "approximate Herbrand theorem" can be obtained: essentially, Herbrand disjunctions exist for successive approximations to validity. We then use this result to show that first-order Łukasiewicz logic admits Skolemization, following the pattern of a similar proof for first-order Gödel logic given in [2].

We define proof calculi for $\forall$**Ł** and its fragments by extending the analytic hypersequent[1] calculus **GŁ** for propositional Łukasiewicz logic introduced in [15]. First we add hypersequent versions of the usual quantifier rules (as e.g. used for first-order Gödel logic [3]) to **GŁ**, obtaining a calculus that with a cut rule is complete with respect to safe MV-algebras. However, as we show with a suitable counter-example, cut-elimination does not hold for this calculus. On the other hand, adding an infinitary rule to the cut-free version of this calculus gives a system that is complete for the full logic. Finally, a finitary cut-free calculus is obtained for the one-variable fragment by relaxing the eigenvariable condition for quantifier rules.

2 Łukasiewicz Logic

First Order Łukasiewicz Logic $\forall$**Ł** is based here on a usual first-order language with connectives $\forall$, $\exists$, $\to$, a constant $\bot$, and defined connectives:

$$\begin{array}{rlcrl} \neg A & =_{def} A \to \bot & \qquad & \top & =_{def} \neg\bot \\ A \oplus B & =_{def} \neg A \to B & & A \odot B & =_{def} \neg(\neg A \oplus \neg B) \\ A \vee B & =_{def} (A \to B) \to B & & A \wedge B & =_{def} \neg(\neg A \vee \neg B) \end{array}$$

For $\# \in \{\wedge, \vee\}$ and a finite multiset of formulas $\Gamma = \{A_1, \dots, A_n\}$ we write $\#\Gamma$ or $\#_{i=1}^{n} A_i$ for $A_1 \# \dots \# A_n$ and let $\wedge\emptyset$ and $\vee\emptyset$ stand for $\top$ and $\bot$ respectively.

Interpretations $\mathcal{I} = (\mathcal{D}, v_{\mathcal{I}})$ for $\forall$**Ł** consist of a non-empty domain $\mathcal{D}$ and a valuation $v_{\mathcal{I}}$ mapping constants and object variables to elements of $\mathcal{D}$; n-ary function symbols to functions from $\mathcal{D}^n$ into $\mathcal{D}$; and n-ary predicate symbols p to functions from $\mathcal{D}^n$ into $[0, 1]$. For atomic formulas:

$$v_{\mathcal{I}}(p(t_1, \dots, t_n)) = v_{\mathcal{I}}(p)(v_{\mathcal{I}}(t_1), \dots, v_{\mathcal{I}}(t_n))$$

For a valuation $v_{\mathcal{I}}$, variable x, and element $a \in \mathcal{D}$, let $v_{\mathcal{I}}[x \leftarrow a]$ be the valuation obtained from $v_{\mathcal{I}}$ by changing $v_{\mathcal{I}}(x)$ to a. Then $v_{\mathcal{I}}$ is extended to all formulas by:

$$\begin{aligned} v_{\mathcal{I}}(\bot) &= 0 \\ v_{\mathcal{I}}(A \to B) &= \min(1, 1 - v_{\mathcal{I}}(A) + v_{\mathcal{I}}(B)) \\ v_{\mathcal{I}}(\forall x A(x)) &= \inf\{v_{\mathcal{I}}[x \leftarrow a](A(x)) : a \in \mathcal{D}\} \\ v_{\mathcal{I}}(\exists x A(x)) &= \sup\{v_{\mathcal{I}}[x \leftarrow a](A(x)) : a \in \mathcal{D}\} \end{aligned}$$

$\mathcal{I}$ *satisfies* a formula A, written $\mathcal{I} \models_{\text{Ł}} A$, iff $v_{\mathcal{I}}(A) = 1$, and A is *valid* in $\forall$**Ł**, written $\models_{\text{Ł}} A$, iff A is satisfied by all $\forall$**Ł** interpretations. Two formulas A and B are $\forall$**Ł**-*equivalent* iff $v_{\mathcal{I}}(A) = v_{\mathcal{I}}(B)$ for all $\forall$**Ł** interpretations $v_{\mathcal{I}}$.

[1] Hypersequents are a generalization of Gentzen sequents introduced by Avron [1], consisting of a multiset (intuitively, a disjunction) of sequents.

It will also be crucial to consider the following notion of "approximate validity" for $\rhd \in \{>, \geq\}$ and $c \in [0,1]$:

$$\models^{\rhd c}_{\text{Ł}} A \text{ iff } v_{\mathcal{I}}(A) \rhd c \text{ for all interpretations } \mathcal{I} \text{ for } \forall\text{Ł}.$$

While the problem of checking validity of formulas in $\forall$**Ł** is Π_2-complete [20], checking approximate validity when $\rhd$ is $>$ and c is rational is Σ_1-complete [5,4]. If the language is restricted to formulas containing at most one variable, i.e. the one-variable fragment, then checking validity becomes decidable [21]. For the monadic fragment, obtained by allowing predicate symbols of arity at most one, satisfiability is Π_1-complete [20] while the complexity of checking validity is an open problem. Finally, propositional Łukasiewicz logic is known to be decidable, indeed co-NP complete [18].

The definition of validity for $\forall$**Ł** given above can be generalized to validity for safe MV-algebras (see [7] for definitions and a wealth of results). An *MV-algebra* is an algebra $\mathcal{A} = \langle L, \oplus, \neg, 0 \rangle$ such that:

1. $\langle L, \oplus, 0 \rangle$ is a commutative monoid.
2. $\neg\neg x = x$.
3. $x \oplus \neg 0 = \neg 0$.
4. $\neg(\neg x \oplus y) \oplus y = \neg(\neg y \oplus x) \oplus x$.

where $1 =_{def} \neg 0$; $x \to y =_{def} \neg x \oplus y$; $x \odot y =_{def} \neg(\neg x \oplus \neg y)$; $x \wedge y =_{def} x \odot (\neg x \oplus y)$; $x \vee y =_{def} (x \odot \neg y) \oplus y$; and $x \leq y$ iff $x \wedge y = y$.

An *MV-chain* is an MV-algebra that is linearly ordered.

An *interpretation* $\mathcal{I}$ for an MV-algebra $\mathcal{A} = \langle L, \oplus, \neg, 0 \rangle$ is defined as for $\forall$**Ł** except that n-ary predicate symbols p are mapped by $v_{\mathcal{I}}$ to functions from $\mathcal{D}^n$ into L and $v_{\mathcal{I}}$ is extended to all formulas by:

$$\begin{aligned}
v_{\mathcal{I}}(\bot) &= 0 \\
v_{\mathcal{I}}(A \to B) &= \begin{cases} v_{\mathcal{I}}(A) \to v_{\mathcal{I}}(B) & \text{if } v_{\mathcal{I}}(A) \text{ and } v_{\mathcal{I}}(B) \text{ are defined} \\ \text{undefined} & \text{otherwise} \end{cases} \\
v_{\mathcal{I}}(\forall x A(x)) &= \begin{cases} \inf\{v_{\mathcal{I}}[x \leftarrow a](A(x)) : a \in \mathcal{D}\} & \text{if the infimum exists} \\ \text{undefined} & \text{otherwise} \end{cases} \\
v_{\mathcal{I}}(\exists x A(x)) &= \begin{cases} \sup\{v_{\mathcal{I}}[x \leftarrow a](A(x)) : a \in \mathcal{D}\} & \text{if the supremum exists} \\ \text{undefined} & \text{otherwise} \end{cases}
\end{aligned}$$

$\mathcal{I}$ *satisfies* a formula A iff $v_{\mathcal{I}}(A) = 1$, and A is *valid* in $\mathcal{A}$ iff A is satisfied by all interpretations for $\mathcal{A}$.

$\mathcal{A}$ is called *safe* if $v_{\mathcal{I}}(A)$ is defined for all formulas A and interpretations $\mathcal{I}$.

The following axiom system **H**$\forall$**Ł** is given by Hájek in [8] (simplifying previous axiomatizations of Hay [10] and Belluce and Chang [5,4]) where $\exists A =_{def} \neg\forall\neg A$:

$$\begin{array}{ll}
(\text{Ł}1) & A \to (B \to A) \\
(\text{Ł}2) & (A \to B) \to ((B \to C) \to (A \to C)) \\
(\text{Ł}3) & ((A \to B) \to B) \to ((B \to A) \to A) \\
(\text{Ł}4) & ((A \to \bot) \to (B \to \bot)) \to (B \to A) \\
(\forall 1) & \forall x A(x) \to A(t) \\
(\forall 2) & \forall x(A \to B) \to (A \to \forall x B) \quad \text{where } x \text{ is not free in } A
\end{array}$$

$$\frac{A \quad A \to B}{B}\ (mp) \qquad\qquad \frac{A}{\forall x A}\ (gen)$$

Theorem 1 ([8]). *The following are equivalent:*

(a) $\vdash_{\mathbf{H}\forall\mathbf{Ł}} A$
(b) A is valid in all safe MV-algebras.
(c) A is valid in all safe MV-chains.

The connection with validity for $\forall\mathbf{Ł}$ is established by the following result:

Theorem 2 ([8]). $\models_{\mathbf{Ł}} A$ *iff* $\vdash_{\mathbf{H}\forall\mathbf{Ł}} A \oplus (\overbrace{A \odot \ldots \odot A}^{n})$ *for all* $1 \leq n \in \mathbb{N}$.

Finally, we recall some key elements of the correspondence between MV-algebras and abelian l-groups (see [17,7] for details).

Lemma 1 ([17,7]). *For every MV-algebra $\mathcal{A}$, there exists an abelian l-group $G = \langle L, +, -, 0\rangle$ with a strong order unit u, $\Xi(\mathcal{A}) = (G, u)$, such that $\mathcal{A}$ is isomorphic to $\langle [0, u], \oplus, \neg, 0\rangle$ where $x \oplus y =_{def} u \wedge (x + y)$ and $\neg x =_{def} u - x$, and the* sup*s and* inf*s of $\mathcal{A}$ coincide with the corresponding* sup*s and* inf*s of* $\langle [0, u], \oplus, \neg, 0\rangle$.

For convenience, we will assume that $\mathcal{A}$ just *is* the algebra $\langle [0, u], \oplus, \neg, 0\rangle$ of $\Xi(\mathcal{A}) = (G, u)$ described in the previous lemma. Also, we will write sums of multisets of elements $x_1 + \ldots + x_n$ in (G, u) as $\sum\{x_1, \ldots, x_n\}$ where $\sum \emptyset = 0$.

3 An Approximate Herbrand Theorem

We begin by recalling some basic notions relating to Herbrand's theorem. Let A be a formula, and let $\mathcal{C}$ and $\mathcal{F}$ be the constants and function symbols occurring in A respectively, adding a constant if the former is empty. The *Herbrand universe* $U(A)$ of A is constructed inductively as follows:

$$\begin{aligned} U_0(A) &= \mathcal{C} \\ U_{n+1}(A) &= U_n(A) \cup \{f(t_1, \ldots, t_k) : t_1, \ldots, t_k \in U_n(A) \text{ and } f \in \mathcal{F} \text{ with arity } k\} \end{aligned}$$

where $U(A) = \bigcup_{n=0}^{\infty} U_n(A)$. The *Herbrand base* $B(A)$ of a formula A is the set of atoms constructed from the predicate symbols of A and the terms occurring in the Herbrand universe $U(A)$. We write $Q\bar{x}A(\bar{x})$ for a formula $Q_1 x_1 \ldots Q_n x_n A(x_1, \ldots, x_n)$ where $Q_i \in \{\forall, \exists\}$ for $1 \leq i \leq n$. As usual a *Prenex formula* is a formula $Q\bar{x}P(\bar{x})$ where P is quantifier-free.

The following useful fact is established easily using admissible quantifier rules:

Theorem 3 ([8]). *Any formula of* $\forall\mathbf{Ł}$ *is* $\forall\mathbf{Ł}$*-equivalent to a Prenex formula of* $\forall\mathbf{Ł}$.

We now observe that Herbrand's Theorem as usually stated does not hold for $\forall\mathbf{Ł}$. First, clearly $\models_{\mathbf{Ł}} \forall x A(x) \to \forall y A(y)$. Hence, using admissible quantifier rules for $\forall\mathbf{Ł}$:

$$\models_{\mathbf{Ł}} \exists y \forall x (A(x) \to A(y))$$

It follows that:

$$\models_{\text{Ł}} \exists y(A(f(y)) \to A(y))$$

So if Herbrand's theorem holds for $\forall$**Ł**, then for some n (where $f^0(a) = a$):

$$\models_{\text{Ł}} \bigvee_{i=1}^{n} (A(f^i(a)) \to A(f^{i-1}(a)))$$

But defining an interpretation $\mathcal{I}$ such that:

$$v_{\mathcal{I}}(A(f^i(a))) > v_{\mathcal{I}}(A(f^{i-1}(a))) \text{ for } i = 1 \ldots n$$

e.g. where $v_{\mathcal{I}}(A(f^i(a))) = i/n$, we obtain a countermodel, so the theorem fails.[2]

Take another look at the formula $\bigvee_{i=1}^{n}(A(f^i(a)) \to A(f^{i-1}(a)))$, however. Although this is not a tautology of $\forall$**Ł**, it comes within "one nth" of being one. That is, we can show that:

$$\models_{\text{Ł}}^{>c} \bigvee_{i=1}^{n} (A(f^{i+1}(a)) \to A(f^i(a)))$$

for any $c < 1 - 1/n$. Hence we can characterize successive "Herbrand approximations" to $\exists y(A(f(y)) \to A(y))$ that come arbitrarily close to 1. Indeed this is an example of a more general phenomenon; captured by the following approximate Herbrand theorem for existential formulas:

Theorem 4. $\models_{\text{Ł}} \exists \bar{x} P(\bar{x})$ *where* P *is quantifier-free iff for all* $c < 1$*:*

$$\models_{\text{Ł}}^{>c} \bigvee_{i=1}^{n} P(\bar{t_i}) \text{ for some } \bar{t_1}, \ldots, \bar{t_n} \in U(P)$$

Proof. If $\models_{\text{Ł}}^{>c} \bigvee_{i=1}^{n} P(\bar{t_i})$ for all $c < 1$, then $\models_{\text{Ł}}^{>c} \exists \bar{x} P(\bar{x})$ for all $c < 1$, so clearly $\models_{\text{Ł}} \exists \bar{x} P(\bar{x})$. For the other direction, fix $c < 1$. Note that each valuation $v_{\mathcal{I}}$ for an interpretation $\mathcal{I}$ may be viewed as a mapping from $B(P)$ into $[0,1]$; i.e. as a member either of $[0,1]^k$ for some k if $B(P)$ is finite, or of the Hilbert cube $[0,1]^\omega$ if $B(P)$ is countably infinite. In either case (the latter using the Tychonoff Theorem), these are compact spaces with respect to the product topology. Now for each $\bar{t} \in U(P)$ define:

$$S(\bar{t}) = \{v_{\mathcal{I}} \in [0,1]^{B(P)} : v_{\mathcal{I}}(P(\bar{t})) \leq c\}$$

Since the connectives of **Ł** are interpreted by continuous functions on $[0,1]$, each $S(\bar{t})$ is a closed subset of $[0,1]^{B(P)}$ with respect to the product topology. We consider:

$$S = \{S(\bar{t}) : \bar{t} \in U(P)\}$$

We have two possibilities:

1. If some $\{S(\bar{t_1}), \ldots, S(\bar{t_n})\} \subseteq S$ has an empty intersection, then for every interpretation $\mathcal{I}$, $v_{\mathcal{I}}(P(\bar{t_i})) > c$ for some i, $1 \leq i \leq n$. I.e. $\models_{\text{Ł}}^{>c} \bigvee_{i=1}^{n} P(\bar{t_i})$ as required.

[2] In fact this argument holds for a wide class of infinite-valued logics.

2. Otherwise, every finite subset of S has a non-empty intersection. Recall that S is a collection of closed subsets of $[0,1]^{B(P)}$. Hence by the finite intersection property for compact spaces, S has a non-empty intersection. There exists $v_{\mathcal{I}}$ such that $v_{\mathcal{I}}(P(\bar{t})) \leq c$ for all $\bar{t} \in U(P)$. So $v_{\mathcal{I}}(\exists \bar{x} P(\bar{x})) \leq c$, a contradiction. □

We obtain as easy consequences the following decidability results.

Proposition 1. *Checking validity for the $\forall\exists$-fragment of $\forall$***Ł** *without function symbols is decidable.*

Proof. A formula $\forall \bar{x} \exists \bar{y} P(\bar{x}, \bar{y})$ where P is quantifier and function-symbol free is valid iff $F = \exists \bar{y} P(\bar{u}, \bar{y})$ is valid for new constants $\bar{u}$. Let $\mathcal{U}$ be the (finite) set of constants occurring in F. By the approximate Herbrand theorem F is valid iff for every $c < 1$, $\models_{\text{Ł}}^{>c} \bigvee_{i=1}^{n} P(\bar{t_i})$ where each $\bar{t_i}$ consists only of constants from $\mathcal{U}$. So F is valid iff the propositional formula $\bigvee \{P(\bar{u}) : \bar{u} \in \mathcal{U}\}$ is valid, a decidable problem. □

Proposition 2. *Checking validity for the one-variable fragment of $\forall$***Ł** *without function symbols is decidable.*

Proof. Any formula in the one-variable fragment of $\forall$**Ł** without function symbols is $\forall$**Ł**-equivalent to a $\forall\exists$ formula without function symbols, so the result follows by the previous proposition. □

4 Skolemization

We will now use the approximate Herbrand theorem in a rather neat way: to obtain Skolemization for $\forall$**Ł**. Recall that the *Skolem form* $\exists \bar{x} A^F(\bar{x})$ of a Prenex formula $Q\bar{y}A(\bar{y})$ is obtained by rewriting $\exists \bar{z} \forall u B(\bar{z}, u)$ to $\exists \bar{z} B(\bar{z}, f(\bar{z}))$ where f is a new function symbol with appropriate arity, as often as possible. Our first step is to show that if a Herbrand disjunction for the Skolem form of a Prenex formula is approximately $\geq c$ valid, then the formula is itself valid to the same degree.

Proposition 3. *Let $\exists \bar{x} A^F(\bar{x})$ be the Skolem form of $Q\bar{y}A(\bar{y})$. If $\models_{\text{Ł}}^{\geq c} \bigvee_{i=1}^{n} A^F(\bar{t_i})$, then $\models_{\text{Ł}}^{\geq c} Q\bar{y}A(\bar{y})$.*

Proof. We assume that the occurrences of $\forall$ in $Q\bar{y}A(\bar{y})$ are labelled with the corresponding function symbols of $\exists \bar{x} A^F(\bar{x})$ introduced by Skolemization. We then define a sequence of sets of formulas as follows. Let $S_0 = \{Q\bar{y}A(\bar{y})\}$. Now, given S_j let S_{j+1} be the smallest set of formulas satisfying:

(1) $S_j \subseteq S_{j+1}$.
(2) If $\forall x B(x) \in S_{j+1}$ and $f(\bar{t})$ labels $\forall$, then $B(f(\bar{t})) \in S_{j+1}$.
(3) If $\exists x B(x) \in S_{j+1}$, then $B'(s) \in S_{j+1}$ for all $s \in U_j(A)$ where B' is B with each $f(\bar{t})$ labelling an occurrence of $\forall$ replaced by $f(\bar{t}, s)$.

Notice that each S_j is finite, since each $U_j(A)$ is finite.

An easy inductive proof shows that for any $\bar{t} \in U(A)$, $A^F(\bar{t}) \in S_k$ for some k: just observe that $\bar{t} \in U_k(A)$ for some k and then use (2) for each occurrence of $\forall$ and (3) for each occurrence of $\exists$ to get that $A^F(\bar{t}) \in S_k$. But this means that if $\models^{\geq c}_{\textbf{Ł}} \bigvee_{i=1}^{n} A^F(\bar{t_i})$, then $\models^{\geq c}_{\textbf{Ł}} \bigvee S_m$ for some large enough m. Hence to complete our proof it is sufficient to show that for each j, if $\models^{\geq c}_{\textbf{Ł}} \bigvee S_{j+1}$, then $\models^{\geq c}_{\textbf{Ł}} \bigvee S_j$. I.e. we have to check that steps (2) and (3) above are sound in the sense that the approximate validity of the extended set implies the approximate validity of the original set:

1. If $\models^{\geq c}_{\textbf{Ł}} \bigvee S \cup \{\forall x B(x), B(d)\}$, then so long as d does not occur in S or B, $\models^{\geq c}_{\textbf{Ł}} \bigvee S \cup \{\forall x B(x)\}$ as required. But the newness of d is guaranteed by the fact that each occurrence of $\forall$ is labelled with a different function symbol f and the arguments of this function are determined uniquely by the terms chosen for the preceding occurrences of $\exists$.
2. If $\models^{\geq c}_{\textbf{Ł}} \bigvee S \cup \{\exists x B(x), B(s)\}$, then $\models^{\geq c}_{\textbf{Ł}} S \cup \{\exists x B(x)\}$ for any term s. □

We can now establish Skolemization for the Prenex fragment of $\forall$**Ł** by combining this last proposition with the approximate Herbrand theorem.

Theorem 5. *Let $\exists \bar{x} A^F(\bar{x})$ be the Skolem form of $Q\bar{y}A(\bar{y})$. Then:*

$$\models_{\textbf{Ł}} \exists \bar{x} A^F(\bar{x}) \ \textit{ iff } \ \models_{\textbf{Ł}} Q\bar{y}A(\bar{y})$$

Proof. The right-to-left direction follows easily using standard quantifier properties of $\forall$**Ł**. For the other direction, suppose that $\models_{\textbf{Ł}} \exists \bar{x} A^F(\bar{x})$. By Theorem 4, for all $c < 1$, there exist tuples of terms, $\bar{t_1}, \ldots, \bar{t_n}$, in $U(A)$ such that $\models^{>c}_{\textbf{Ł}} \bigvee_{i=1}^{n} A^F(\bar{t_i})$. But then by Proposition 3, for all $c < 1$, $\models^{\geq c}_{\textbf{Ł}} Q\bar{y}A(\bar{y})$. Hence $\models_{\textbf{Ł}} Q\bar{y}A(\bar{y})$ as required. □

Since by Theorem 3, any formula has an equivalent Prenex formula, we obtain:

Corollary 1. *Skolemization holds for $\forall$**Ł**.*

Note that (the easy direction of) Skolemization allows us to extend the approximate Herbrand theorem to the whole of $\forall$**Ł**: we just put the formula into Prenex form, apply Theorem 5 and then Theorem 4.

Corollary 2. *Let A be a formula and let $\exists \bar{x} P^F(\bar{x})$ be the Skolem form of an equivalent Prenex formula for A. Then $\models_{\textbf{Ł}} A$ iff for all $c < 1$:*

$$\models^{>c}_{\textbf{Ł}} \bigvee_{i=1}^{n} P^F(\bar{t_i}) \ \textit{ for some } \bar{t_1}, \ldots, \bar{t_n} \in U(P^F)$$

5 The Hypersequent Calculus GŁ

We define proof systems for Łukasiewicz logic in the framework of *hypersequents*; finite multisets of the form:

$$\Gamma_1 \Rightarrow \Delta_1 \mid \ldots \mid \Gamma_n \Rightarrow \Delta_n$$

where Γ_i and Δ_i are finite multisets of formulas for $i = 1 \ldots n$; i.e. each $\Gamma_i \Rightarrow \Delta_i$ is an ordinary sequent, called a *component* of the hypersequent.

Validity is extended to hypersequents as follows:

Definition 1. $\models_{\mathbf{Ł}} \mathit{\Gamma}_1 \Rightarrow \mathit{\Delta}_1 \mid \ldots \mid \mathit{\Gamma}_n \Rightarrow \mathit{\Delta}_n$ *iff for all* $\forall\mathbf{Ł}$ *interpretations* $\mathcal{I}$*:*

$$\sum\{v_{\mathcal{I}}(A) - 1 : A \in \mathit{\Gamma}_i\} \leq \sum\{v_{\mathcal{I}}(B) - 1 : B \in \mathit{\Delta}_i\} \text{ for some } i,\ 1 \leq i \leq n.$$

More generally, we can define validity for hypersequents for any safe MV-algebra.

Definition 2. $\mathit{\Gamma}_1 \Rightarrow \mathit{\Delta}_1 \mid \ldots \mid \mathit{\Gamma}_n \Rightarrow \mathit{\Delta}_n$ *is valid in a safe MV-algbera* $\mathcal{A}$ *iff for all interpretations* $\mathcal{I}$ *for* $\mathcal{A}$*:*

$$\sum\{v_{\mathcal{I}}(A) - u : A \in \mathit{\Gamma}_i\} \leq \sum\{v_{\mathcal{I}}(B) - u : B \in \mathit{\Delta}_i\} \text{ for some } i,\ 1 \leq i \leq n.$$

in $\Xi(\mathcal{A}) = (G, u)$*, the abelian l-group* $G = \langle L, +, -, 0\rangle$ *with order unit* u*.*

These interpretations are certainly not standard. Hypersequents are interpreted using sums of elements in abelian l-groups rather than as formulas of **Ł**. In particular, the left and right comma are not interpreted as the $\odot$ and $\oplus$ of an MV-algebra, but “outside the logic” as the addition $+$ of the corresponding abelian l-group. Nevertheless, for formulas we still obtain the usual notion of validity for **Ł**, i.e. we have that a formula A is valid iff $\models_{\mathbf{Ł}} \Rightarrow A$ (for propositional formulas, iff $\vdash_{\mathbf{H\forall Ł}} \Rightarrow A$).

The following analytic (i.e. having the subformula property and no cut rule) hypersequent calculus for propositional Łukasiewicz logic was introduced in [15]. As usual, we write $\mathit{\Gamma}, \mathit{\Pi}$ for the multiset union of $\mathit{\Gamma}$ and $\mathit{\Pi}$ and $\mathit{\Gamma}, A$ for the multiset union of $\mathit{\Gamma}$ and $\{A\}$. We also use $\mathcal{G}$ to denote an arbitrary “side-hypersequent” occurring in both the premises and conclusion of a rule.

Definition 3 (The Hypersequent Calculus GŁ)

Initial Sequents:

$$\frac{}{A \Rightarrow A}\,(id) \qquad \frac{}{\Rightarrow}\,(\Lambda) \qquad \frac{}{\bot \Rightarrow A}\,(\bot\Rightarrow)$$

Structural Rules:

$$\frac{\mathcal{G}}{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow \mathit{\Delta}}\,(ew) \qquad \frac{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow \mathit{\Delta} \mid \mathit{\Gamma} \Rightarrow \mathit{\Delta}}{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow \mathit{\Delta}}\,(ec) \qquad \frac{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow \mathit{\Delta}}{\mathcal{G} \mid \mathit{\Gamma}, A \Rightarrow \mathit{\Delta}}\,(wl)$$

$$\frac{\mathcal{G} \mid \mathit{\Gamma}_1, \mathit{\Gamma}_2 \Rightarrow \mathit{\Delta}_1, \mathit{\Delta}_2}{\mathcal{G} \mid \mathit{\Gamma}_1 \Rightarrow \mathit{\Delta}_1 \mid \mathit{\Gamma}_2 \Rightarrow \mathit{\Delta}_2}\,(split) \qquad \frac{\mathcal{G} \mid \mathit{\Gamma}_1 \Rightarrow \mathit{\Delta}_1 \quad \mathcal{G} \mid \mathit{\Gamma}_2 \Rightarrow \mathit{\Delta}_2}{\mathcal{G} \mid \mathit{\Gamma}_1, \mathit{\Gamma}_2 \Rightarrow \mathit{\Delta}_1, \mathit{\Delta}_2}\,(mix)$$

Logical Rules:

$$\frac{\mathcal{G} \mid \mathit{\Gamma}, B \Rightarrow A, \mathit{\Delta}}{\mathcal{G} \mid \mathit{\Gamma}, A \rightarrow B \Rightarrow \mathit{\Delta}}\,(\rightarrow\Rightarrow) \qquad \frac{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow \mathit{\Delta} \quad \mathcal{G} \mid \mathit{\Gamma}, A \Rightarrow B, \mathit{\Delta}}{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow A \rightarrow B, \mathit{\Delta}}\,(\Rightarrow\rightarrow)$$

Rules for defined connectives can also be obtained, e.g.

$$\frac{\mathcal{G} \mid \mathit{\Gamma}, A \Rightarrow \mathit{\Delta}}{\mathcal{G} \mid \mathit{\Gamma}, A \wedge B \Rightarrow \mathit{\Delta}}\,(\wedge\Rightarrow)_1 \qquad \frac{\mathcal{G} \mid \mathit{\Gamma}, B \Rightarrow \mathit{\Delta}}{\mathcal{G} \mid \mathit{\Gamma}, A \wedge B \Rightarrow \mathit{\Delta}}\,(\wedge\Rightarrow)_2$$

$$\frac{\mathcal{G} \mid \mathit{\Gamma}, A \Rightarrow \mathit{\Delta} \quad \mathcal{G} \mid \mathit{\Gamma}, B \Rightarrow \mathit{\Delta}}{\mathcal{G} \mid \mathit{\Gamma}, A \vee B \Rightarrow \mathit{\Delta}}\,(\vee\Rightarrow) \qquad \frac{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow A, \mathit{\Delta} \quad \mathcal{G} \mid \mathit{\Gamma} \Rightarrow B, \mathit{\Delta}}{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow A \wedge B, \mathit{\Delta}}\,(\Rightarrow\wedge)$$

$$\frac{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow A, \mathit{\Delta}}{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow A \vee B, \mathit{\Delta}}\,(\Rightarrow\vee)_1 \qquad \frac{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow B, \mathit{\Delta}}{\mathcal{G} \mid \mathit{\Gamma} \Rightarrow A \vee B, \mathit{\Delta}}\,(\Rightarrow\vee)_2$$

Note moreover that a simpler ("standard") version of the implication right rule is derivable when only one formula appears on the right:

$$\dfrac{\mathcal{G} \mid \Gamma, A \Rightarrow B}{\mathcal{G} \mid \Gamma \Rightarrow A \to B}\ (\Rightarrow\to)_1$$

Example 1. We illustrate this calculus with a derivation of the key axiom (Ł3):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B \Rightarrow B}\,(id) \quad \dfrac{}{A \Rightarrow A}\,(id)}{B, A \Rightarrow A, B}\,(mix)}{B, B \to A \Rightarrow A}\,(\to\Rightarrow) \quad \dfrac{\dfrac{\dfrac{}{B \Rightarrow B}\,(id) \quad \dfrac{}{A \Rightarrow A}\,(id)}{B, A \Rightarrow A, B}\,(mix)}{B, B \to A, A \Rightarrow A, B}\,(wl)}{B, B \to A \Rightarrow A, A \to B}\,(\Rightarrow\to)}{(A \to B) \to B, B \to A \Rightarrow A}\,(\to\Rightarrow)}{(A \to B) \to B \Rightarrow (B \to A) \to A}\,(\Rightarrow\to)_1}{\Rightarrow ((A \to B) \to B) \to ((B \to A) \to A)}\,(\Rightarrow\to)_1$$

Observe that hypersequents are not needed to prove this or indeed any of the other propositional axioms (Ł1)-(Ł4) for **Ł**; nevertheless, they are essential to prove other theorems such as $A \to (B \to ((A \to (A \to C)) \to ((B \to (B \to C)) \to C)))$.

A semantic completeness proof for **GŁ** was presented in [15].

Theorem 6 ([15]). $\vdash_{\mathbf{G\text{Ł}}} \mathcal{G}$ *iff* $\models_{\mathbf{\text{Ł}}} \mathcal{G}$.

It follows from this theorem that the following cut rule and (inter-derivable) cancellation rule are admissible for **GŁ**:

$$\dfrac{\mathcal{G} \mid \Gamma_1, A \Rightarrow \Delta_1 \quad \mathcal{G} \mid \Gamma_2 \Rightarrow A, \Delta_2}{\mathcal{G} \mid \Gamma_1, \Gamma_2 \Rightarrow \Delta_1, \Delta_2}\ (cut) \qquad \dfrac{\mathcal{G} \mid \Gamma, A \Rightarrow A, \Delta}{\mathcal{G} \mid \Gamma \Rightarrow \Delta}\ (can)$$

Syntactic eliminations of these rules were given in [6].

Theorem 7 ([6]). *Cut-elimination holds for* **GŁ** + (cut) *and cancellation-elimination holds for* **GŁ** + (can).

6 Adding Quantifiers

In this section we consider the effect of adding the obvious "standard quantifier rules" to **GŁ**; i.e. adding the rules used for other first-order fuzzy logics such as $\forall$**G** [3]: the hypersequent versions of Gentzen's rules for **LJ** and **LK**.

Definition 4. **G$\forall$Ł** *is* **GŁ** *extended with:*

$$\dfrac{\mathcal{G} \mid \Gamma, A(t) \Rightarrow \Delta}{\mathcal{G} \mid \Gamma, \forall x A(x) \Rightarrow \Delta}\ (\forall\Rightarrow) \qquad \dfrac{\mathcal{G} \mid \Gamma \Rightarrow A(a), \Delta}{\mathcal{G} \mid \Gamma \Rightarrow \forall x A(x), \Delta}\ (\Rightarrow\forall)$$

$$\dfrac{\mathcal{G} \mid \Gamma, A(a) \Rightarrow \Delta}{\mathcal{G} \mid \Gamma, \exists x A(x) \Rightarrow \Delta}\ (\exists\Rightarrow) \qquad \dfrac{\mathcal{G} \mid \Gamma \Rightarrow A(t), \Delta}{\mathcal{G} \mid \Gamma \Rightarrow \exists x A(x), \Delta}\ (\Rightarrow\exists)$$

where a does not occur in the conclusion of $(\Rightarrow\forall)$ *or* $(\exists\Rightarrow)$.

Example 2. Consider the following proof in $\mathbf{G\forall Ł}$ of the axiom $(\forall 2)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B(a)\Rightarrow B(a)}\,(id) \qquad \dfrac{}{A\Rightarrow A}\,(id)}{B(a),A\Rightarrow A,B(a)}\,(mix)}{A\to B(a),A\Rightarrow B(a)}\,(\to\Rightarrow)}{\forall x(A\to B),A\Rightarrow B(a)}\,(\forall\Rightarrow)}{\forall x(A\to B),A\Rightarrow \forall xB}\,(\Rightarrow\forall)}{\forall x(A\to B)\Rightarrow A\to\forall xB}\,(\Rightarrow\to)_1}{\Rightarrow \forall x(A\to B)\to(A\to\forall xB)}\,(\Rightarrow\to)_1$$

Notice that it is crucial for this proof that x does not occur free in A, since in that case $A(a)$ would appear on the left in the second application of (id), with $A(x)$ on the right.

It is easy to check that this system is sound with respect to $\models_{\mathbf{Ł}}$.

Theorem 8. *If* $\vdash_{\mathbf{G\forall Ł}} \mathcal{G}$*, then* $\models_{\mathbf{Ł}} \mathcal{G}$.

Proof. By induction on the height of a derivation of $\mathcal{G}$ in $\mathbf{G\forall Ł}$. Since all other rules have been checked in [15], we just check that the quantifier rules preserve validity. Also note that using linearity, we can disregard the side-hypersequent $\mathcal{G}$ occurring in both premises and conclusion. For $(\Rightarrow\forall)$, suppose that for each interpretation $\mathcal{I}$:

$$\sum_{C\in\Gamma}(v_{\mathcal{I}}(C)-1)\leq(v_{\mathcal{I}}(A(a))-1)+\sum_{D\in\Delta}(v_{\mathcal{I}}(D)-1)$$

where a does not occur in Γ, Δ, or A. Since this holds for any a, we have also:

$$\sum_{C\in\Gamma}(v_{\mathcal{I}}(C)-1)\leq(v_{\mathcal{I}}(\forall xA(x))-1)+\sum_{D\in\Delta}(v_{\mathcal{I}}(D)-1)$$

as required. For $(\forall\Rightarrow)$, suppose that:

$$\sum_{C\in\Gamma}(v_{\mathcal{I}}(C)-1)+(v_{\mathcal{I}}(A(t))-1)\leq\sum_{D\in\Delta}(v_{\mathcal{I}}(D)-1)$$

for some term t, then:

$$\sum_{C\in\Gamma}(v_{\mathcal{I}}(C)-1)+(v_{\mathcal{I}}(\forall xA(x))-1)\leq\sum_{D\in\Delta}(v_{\mathcal{I}}(D)-1)$$

The cases of the rules for $\exists$ are very similar. □

Moreover, if we extend $\mathbf{G\forall Ł}$ with (cut) or (can), then we obtain completeness with respect to the axiomatization $\mathbf{H\forall Ł}$, noting that completeness for $\forall\mathbf{Ł}$ cannot be possible for any finitary calculus since the logic is not recursively axiomatizable.

Theorem 9. $\vdash_{\mathbf{H\forall Ł}} A$ *iff* $\vdash_{\mathbf{G\forall Ł}+(cut)}\Rightarrow A$.

Proof. For the left-to-right direction, we observe that (gen) and (mp) are admissible for $\mathbf{G\forall Ł}+(cut)$ using $(\Rightarrow\forall)$ and (cut), respectively. Moreover, it is easy to see that the axioms (Ł1)-(Ł4) and ($\forall$1)-($\forall$2) are derivable. For the right-to-left direction, we can prove the more general claim: if $\vdash_{\mathbf{G\forall Ł}+(cut)} \mathcal{G}$, then $\vdash_{\mathbf{H\forall Ł}} \mathcal{G}$, proceeding by induction on the height of a derivation of $\mathcal{G}$ in **G∀Ł**. We check the soundness of each rule by restricting to safe MV-chains and recalling that for each safe MV-algebra $\mathcal{A}$ and abelian l-group G with strong unit u such that $\Xi(\mathcal{A}) = (G, u)$, we have that $\mathcal{A}$ is (isomorphic to) $\langle [0, u], \oplus, \neg, 0\rangle$ where $x \oplus y =_{def} u \wedge (x + y)$ and $\neg x =_{def} u - x$. As an example, consider $(\rightarrow\Rightarrow)$. Suppose that for some interpretation $\mathcal{I}$:

$$\sum_{C\in\Gamma}(v_{\mathcal{I}}(C) - u) + (v_{\mathcal{I}}(B) - u) \leq \sum_{D\in\Delta}(v_{\mathcal{I}}(D) - u) + (v_{\mathcal{I}}(A) - u)$$

It follows that:

$$\sum_{C\in\Gamma}(v_{\mathcal{I}}(C) - u) + ((u - v_{\mathcal{I}}(A) + v_{\mathcal{I}}(B)) - u) \leq \sum_{D\in\Delta}(v_{\mathcal{I}}(D) - u)$$

Hence, since also $(u - v_{\mathcal{I}}(A) + v_{\mathcal{I}}(B)) \leq u$, we get:

$$\sum_{C\in\Gamma}(v_{\mathcal{I}}(C) - u) + (v_{\mathcal{I}}(A \rightarrow B) - u) \leq \sum_{D\in\Delta}(v_{\mathcal{I}}(D) - u)$$

as required. Other cases are similar. □

However, cut elimination fails for **G∀Ł** + (cut) and **G∀Ł** + (can). For example, $\exists x\forall y(A(x) \rightarrow A(y))$ has the following proof in **G∀Ł** + (can):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{A(a) \Rightarrow A(a)}\;(id)}{\forall z A(z) \Rightarrow A(a)}\;(\forall\Rightarrow) \qquad \dfrac{\overline{A(a) \Rightarrow A(a)}\;(id) \qquad \dfrac{\overline{A(b) \Rightarrow A(b)}\;(id)}{\forall z A(z) \Rightarrow A(b)}\;(\forall\Rightarrow)}{\forall z A(z), A(a) \Rightarrow A(b), A(a)}\;(mix)}{\forall z A(z) \Rightarrow A(a) \rightarrow A(b), A(a)}\;(\Rightarrow\rightarrow)}{\forall z A(z) \Rightarrow \forall y(A(a) \rightarrow A(y)), A(a)}\;(\Rightarrow\forall)}{\forall z A(z) \Rightarrow \exists x\forall y(A(x) \rightarrow A(y)), A(a)}\;(\Rightarrow\exists)}{\forall z A(z) \Rightarrow \exists x\forall y(A(x) \rightarrow A(y)), \forall z A(z)}\;(\Rightarrow\forall)}{\Rightarrow \exists x\forall y(A(x) \rightarrow A(y))}\;(can)$$

But this formula has no proof in **G∀Ł**.

7 An Infinitary Calculus

To obtain a calculus that is complete for ∀**Ł**, we need (as in the axiomatizations of Hay [10], Belluce and Chang [5,4], and Hájek [8]) an infinitary rule. In particular, we can express our required rule in (hyper)sequent format as:

$$\dfrac{\Rightarrow A \oplus (\overbrace{A \odot \ldots \odot A}^{n}) \;\text{ for all } 1 \leq n \in \mathbb{N}}{\Rightarrow A}$$

Alternatively, writing A^n for the multiset containing n copies of A, we could use:

$$\frac{\bot \Rightarrow A^n \text{ for all } 1 \leq n \in \mathbb{N}}{\Rightarrow A}$$

or introduce rational constants into the language of the form $\bar{r}$ for each $r \in [0,1] \cap \mathbb{Q}$ (requiring of course further book-keeping axioms and rules) and use:

$$\frac{\overline{1 - 1/n} \Rightarrow A \text{ for all } 1 \leq n \in \mathbb{N}}{\Rightarrow A}$$

To establish the completeness of **G∀Ł** extended with such rules we proceed in similar fashion to the proof of Proposition 3, the complicating factors here being the presence of quantifiers deep within the formula and the use of hypersequent rules to decompose the formula into sets of hypersequents.

Proposition 4. *Let $Q\bar{y}A(\bar{y})$ be a Prenex form of a formula C with Skolem form $\exists\bar{x}A^F(\bar{x})$. If $\models_{\text{Ł}} \bigvee_{i=1}^{n} A^F(\bar{t}_i)$, then $\vdash_{\mathbf{G\forall Ł}} \Rightarrow C$.*

Proof. Let the sequence S_j of sets of formulas with labelled occurrences be defined for $Q\bar{y}A(\bar{y})$ exactly as in Proposition 3 and label with the same function symbols the corresponding occurrences of $\forall$ and $\exists$ in C (noting that some occurrences of $\exists$ are transformed to $\forall$ while prenexing and vice versa). Now we define sets H_j of hypersequents that contain only terms from $U_j(A)$ as follows. Let $H_0 = \{\Rightarrow C\}$ and given H_j, let H_{j+1} be the result of applying the following operations to H_j as many times as possible (it is easy to see that this process terminates):

- Replace $\mathcal{G} \mid \Gamma, A \to B \Rightarrow \Delta$ with $\mathcal{G} \mid \Gamma \Rightarrow \Delta \mid \Gamma, B \Rightarrow A, \Delta$.
- Replace $\mathcal{G} \mid \Gamma \Rightarrow A \to B, \Delta$ with $\mathcal{G} \mid \Gamma \Rightarrow \Delta$ and $\mathcal{G} \mid \Gamma, A \Rightarrow B, \Delta$.
- If $\mathcal{G} \mid \Gamma \Rightarrow \forall x B(x), \Delta$ occurs in the set and $f(\bar{t})$ labels this occurrence of $\forall$, then add $\mathcal{G} \mid \Gamma \Rightarrow B(f(\bar{t})), \Delta$.
- If $\mathcal{G} \mid \Gamma, \forall x B(x) \Rightarrow \Delta$ occurs in the set, then add $\mathcal{G} \mid \Gamma, B'(s) \Rightarrow \Delta$ for all $s \in U_j(A)$ where B' is B with each $f(\bar{t})$ labelling a positive occurrence of $\forall$ or negative occurrence of $\exists$ replaced by $f(\bar{t}, s)$.
- If $\mathcal{G} \mid \Gamma, \exists x B(x) \Rightarrow \Delta$ occurs in the set and $f(\bar{t})$ labels this occurrence of $\exists$, then add $\mathcal{G} \mid \Gamma, B(f(\bar{t})) \Rightarrow \Delta$.
- If $\mathcal{G} \mid \Gamma \Rightarrow \exists x B(x), \Delta$ occurs in the set, then add $\mathcal{G} \mid \Gamma \Rightarrow B'(s), \Delta$ for all $s \in U_j(A)$ where B' is B with each $f(\bar{t})$ labelling a positive occurrence of $\forall$ or negative occurrence of $\exists$ replaced by $f(\bar{t}, s)$.

Observe that using the implication, quantifier, weakening, and external contraction rules of **G∀Ł**, we have that for each j, if $\vdash_{\mathbf{G\forall Ł}} \mathcal{G}$ for all $\mathcal{G} \in H_{j+1}$, then $\vdash_{\mathbf{G\forall Ł}} \mathcal{G}$ for all $\mathcal{G} \in H_j$. Hence it is sufficient to show that $\vdash_{\mathbf{G\forall Ł}} \mathcal{G}$ for all $\mathcal{G} \in H_k$ for some k. To show this, we define:

$$\mathit{prop}(X) = \{S \in X : \text{S contains only propositional formulas}\}$$

Recall that each S_j is a set of formulas containing only terms from $U_j(A)$. It follows from the construction of the two sequences that if $\models_{\text{Ł}} \bigvee \mathit{prop}(S_j)$, then also

$\models_{\mathbf{Ł}} prop(\mathcal{G})$ for each $\mathcal{G} \in H_j$. Hence by the completeness of **GŁ**, $prop(\mathcal{G})$ is derivable in **GŁ** from $\Rightarrow \bigvee prop(S_j)$ for each $\mathcal{G} \in H_j$. But now for some k, $\models_{\mathbf{Ł}} \bigvee prop(S_k)$. It follows that $\vdash_{\mathbf{GŁ}} \mathcal{G}$ for each $\mathcal{G} \in H_k$ as required. □

Theorem 10. *$\models_{\mathbf{Ł}} A$ iff $\vdash_{\mathbf{G\forall Ł}} \Rightarrow A \oplus (\overbrace{A \odot \ldots \odot A}^{n})$ for all $1 \leq n \in \mathbb{N}$.*

Proof. For the right-to-left direction, observe that if $\vdash_{\mathbf{G\forall Ł}} A \oplus (\overbrace{A \odot \ldots \odot A}^{n})$ for all $1 \leq n \in \mathbb{N}$, then in all interpretations $\mathcal{I}$, $\frac{n-1}{n} \leq v_{\mathcal{I}}(A)$, for all $1 \leq n \in \mathbb{N}$, i.e. $v_{\mathcal{I}}(A) = 1$ for all interpretations $\mathcal{I}$. For the left-to-right direction suppose that $\models_{\mathbf{Ł}} A$. Let $Q\bar{y}P(y)$ be a Prenex form of A and let $\exists \bar{x} P^F(\bar{x})$ be the Skolem form of $Q\bar{y}P(y)$. By Theorem 5, $\models_{\mathbf{Ł}} \exists \bar{x} P^F(\bar{x})$, so by Theorem 4, for all $1 \leq n \in \mathbb{N}$:

$$\models_{\mathbf{Ł}}^{>1-1/n} \bigvee_{i=1}^{m} P^F(\bar{t_i}) \text{ for some } \bar{t_1}, \ldots, \bar{t_m} \in U(P)$$

Hence, reasoning in the standard model, for all $1 \leq n \in \mathbb{N}$:

$$\models_{\mathbf{Ł}} \bigvee_{i=1}^{m} (P_1^F \oplus (\overbrace{P_2^F \odot \ldots \odot P_n^F}^{n}))(\bar{t_i}) \text{ for some } \bar{t_1}, \ldots, \bar{t_m} \in U(P)$$

where each P_j^F is obtained by replacing each Skolem function f in P^F with a distinguished Skolem function f_j. But then by the previous proposition $\vdash_{\mathbf{G\forall Ł}} \Rightarrow A \oplus (\overbrace{A \odot \ldots \odot A}^{n})$ for all $1 \leq n \in \mathbb{N}$ as required. □

This completeness result transfers easily to other systems, e.g.:

Corollary 3. *$\models_{\mathbf{Ł}} A$ iff $\Rightarrow A$ is derivable in $\mathbf{G\forall Ł}$ extended with the rule:*

$$\frac{\bot \Rightarrow nA \;\; \textit{for all } 1 \leq n \in \mathbb{N}}{\Rightarrow A}$$

Example 3. Consider our earlier problematic formula $\exists x \forall y(A(x) \to A(y))$. To prove this in the above system, we would have to perform an infinite number of derivations in **G∀Ł**. E.g in the case where $n = 2$, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\bot \Rightarrow A(b) \to A(c)}\,(\bot\Rightarrow) \quad \dfrac{\dfrac{\dfrac{}{\bot \Rightarrow A(b)}\,(\bot\Rightarrow) \quad \dfrac{\dfrac{}{\bot \Rightarrow A(c)}\,(\bot\Rightarrow) \quad \dfrac{}{A(b) \Rightarrow A(b)}\,(id)}{\bot, A(b) \Rightarrow A(b), A(c)}\,(mix)}{\bot \Rightarrow A(b), A(b) \to A(c)}\,(\Rightarrow\to)}{\bot, A(a) \Rightarrow A(b), A(b) \to A(c)}\,(wl)}{\bot \Rightarrow A(a) \to A(b), A(b) \to A(c)}\,(\Rightarrow\to)}{\bot \Rightarrow A(a) \to A(b), \forall y(A(b) \to A(y))}\,(\Rightarrow\forall)}{\bot \Rightarrow A(a) \to A(b), \exists x \forall y(A(x) \to A(y))}\,(\Rightarrow\exists)}{\bot \Rightarrow \forall y(A(a) \to A(y)), \exists x \forall y(A(x) \to A(y))}\,(\Rightarrow\forall)}{\bot \Rightarrow \exists x \forall y(A(x) \to A(y)), \exists x \forall y(A(x) \to A(y))}\,(\Rightarrow\exists)$$

8 The One Variable Fragment

As established in Section 3, the decidability of the one variable fragment of $\forall$**Ł** without function symbols follows from the approximate Herbrand theorem for $\forall$**Ł**. Here, we provide a hypersequent calculus for this fragment by liberalising the eigenvariable condition for $(\Rightarrow\forall)$ and $(\exists\Rightarrow)$: the idea being that we allow for quantifier shifts that "could have been performed earlier in the proof".

Definition 5. *Let* $\mathbf{G}\forall\mathbf{Ł_1}$ *be* $\mathbf{G}\forall\mathbf{Ł}$ *with the eigenvariable condition changed to "a is either new or removed by* $(\forall\Rightarrow)$ *or* $(\Rightarrow\exists)$ *at a lower point in the proof".*

The new eigenvariable condition given here is global in the sense that it applies to whole proofs: indeed the rules are not sound in isolation, only as part of a proof.

Example 4. Consider the following proof of $\exists x(A(x) \to \forall x A(x))$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A(a) \Rightarrow A(a)}\,(id)}{A(a) \Rightarrow \forall x A(x)}\,(\Rightarrow\forall)}{\Rightarrow A(a) \to \forall x A(x)}\,(\Rightarrow\to)_1}{\Rightarrow \exists x(A(x) \to \forall x A(x))}\,(\Rightarrow\exists)$$

Notice that the use of the constant a in the application of $(\Rightarrow\forall)$ is justified by the fact that a is removed by $(\Rightarrow\exists)$ two lines down in the proof. In fact, the subproof ending with $A(a) \Rightarrow \forall x A(x)$ is not allowed in isolation: rightly so, since the sequent is not valid for $\forall$**Ł**.

Theorem 11. *For each one-variable formula A, $\vdash_{\mathbf{G}\forall\mathbf{Ł_1}} \Rightarrow A$ iff $\models_{\mathbf{Ł}} A$.*

Proof. Let A' be A with all positive occurrences of $\forall x B(x)$ and negative occurrences of $\exists x B(x)$ replaced by $B(a)$ where a is a new variable in each case. Since A only has one variable, it follows using admissible quantifier rules for $\forall$**Ł** that $\models_{\mathbf{Ł}} A'$ iff $\models_{\mathbf{Ł}} A$. For soundness, if $\vdash_{\mathbf{G}\forall\mathbf{Ł_1}} \Rightarrow A$, then by an easy induction $\vdash_{\mathbf{G}\forall\mathbf{Ł}} \Rightarrow A'$. Hence by the soundness of $\mathbf{G}\forall\mathbf{Ł}$, $\models_{\mathbf{Ł}} A'$ and therefore also $\models_{\mathbf{Ł}} A$.

For completeness, suppose that $\models_{\mathbf{Ł}} A$ and let the Prenex form of A' be $\exists\bar{x}P(\bar{x})$ where P is quantifier-free. Since $\models_{\mathbf{Ł}} \exists\bar{x}P(\bar{x})$, by the approximate Herbrand theorem, $\models_{\mathbf{Ł}} P_1(\bar{t_1}) \vee \ldots \vee P_n(\bar{t_n})$ for some $\bar{t_1}, \ldots, \bar{t_n}$. By Theorem 6, $\vdash_{\mathbf{GŁ}} \Rightarrow P_1(\bar{t_1}) \mid \ldots \mid \Rightarrow P_n(\bar{t_n})$. Hence to prove $\Rightarrow A$ we apply (ec) upwards n times to obtain the hypersequent $\Rightarrow A \mid \ldots \mid \Rightarrow A$. We then mimic the proof of $\Rightarrow P_1(\bar{t_1}) \mid \ldots \mid \Rightarrow P_n(\bar{t_n})$ making sure that we choose the corresponding terms when we encounter occurrences of $\forall$ and $\exists$. For $(\forall\Rightarrow)$ and $(\Rightarrow\exists)$, this is fine since we can choose terms as we like. The only problem that can occur is in the rules $(\Rightarrow\forall)$ and $(\exists\Rightarrow)$ but the relaxed eigenvariable condition takes care of this: any variable that we need is either new or removed further down the proof by $(\forall\Rightarrow)$ or $(\Rightarrow\exists)$. □

Concluding Remark. While the main achievements of this paper are theoretical, the approximate Herbrand theorem and Skolemization results should be useful in investigating fragments of practical use. In particular, we would like to consider decidable fragments for fuzzy description logics (as described in e.g. [9,23]) and fuzzy logic programming (see e.g. [24]). It is our hope also that our approach will shed some light on the intriguing problem of the decidability of monadic Łukasiewicz logic.

References

1. Avron, A.: A constructive analysis of RM. Journal of Symbolic Logic 52(4), 939–951 (1987)
2. Baaz, M., Ciabattoni, A., Fermüller, C.G.: Herbrand's theorem for Prenex Gödel logic and its consequences for theorem proving. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, Springer, Heidelberg (2001)
3. Baaz, M., Zach, R.: Hypersequents and the proof theory of intuitionistic fuzzy logic. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, Springer, Heidelberg (2000)
4. Belluce, L.P.: Further results on infinite valued predicate logic. Journal of Symbolic Logic 29, 69–78 (1964)
5. Belluce, L.P., Chang, C.C.: A weak completeness theorem for infinite valued first order logic. Journal of Symbolic Logic 28, 43–50 (1963)
6. Ciabattoni, A., Metcalfe, G.: Bounded Łukasiewicz logics. In: Mayer, M.C., Pirri, F. (eds.) TABLEAUX 2003. LNCS, vol. 2796, Springer, Heidelberg (2003)
7. Cignoli, R., D'Ottaviano, I.M.L., Mundici, D.: Algebraic Foundations of Many-Valued Reasoning, volume 7 of Trends in Logic. Kluwer, Dordrecht (1999)
8. Hájek, P.: Metamathematics of Fuzzy Logic. Kluwer, Dordrecht (1998)
9. Hájek, P.: Making fuzzy description logic more general. Fuzzy Sets. and Systems 23(2–4), 161–178 (2006)
10. Hay, L.S.: Axiomatization of the infinite-valued predicate calculus. Journal of Symbolic Logic 28(1), 77–86 (1963)
11. Łukasiewicz, J., Łukasiewicz, Jan.: Selected Writings. North-Holland, Edited by L. Borowski (1970)
12. Łukasiewicz, J., Tarski, A.: Untersuchungen über den Aussagenkalkül. Comptes Rendus des Séances de la Societé des Sciences et des Lettres de Varsovie, Classe III, 23, Reprinted and translated in (1930)
13. Malinowski, G.: Many-Valued Logics. Oxford Logic Guides, vol. 25. Oxford University Press, New York (1993)
14. McNaughton, R.: A theorem about infinite-valued sentential logic. Journal of Symbolic Logic 16(1), 1–13 (1951)
15. Metcalfe, G., Olivetti, N., Gabbay, D.: Sequent and hypersequent calculi for abelian and Łukasiewicz logics. ACM Transactions on Computational Logic 6(3), 578–613 (2005)
16. Mostowski, A.: Axiomatizability of some many valued predicate calculi. Fundamentica Mathematica 50, 165–190 (1961)
17. Mundici, D.: Interpretation of AF C*-algebras in Łukasiewicz sentential calculus. Journal of Functional Analysis 65, 15–63 (1986)
18. Mundici, D.: Satisfiability in many-valued sentential logic is NP-complete. Theoretical Computer Science 52(1-2), 145–153 (1987)
19. Novak, V.: On the Hilbert-Ackermann theorem in fuzzy logic. Acta. Mathematica et Informatica Universitatis Ostraviensis 4, 57–74 (1996)
20. Ragaz, M.E.: Arithmetische Klassifikation von Formelmengen der unendlichwertigen Logik. PhD thesis, ETH Zürich (1981)
21. Rutledge, J.D.: A preliminary investigation of the infinitely many-valued predicate calculus. PhD thesis, Cornell University, Ithaca (1959)
22. Scarpellini, B.: Die Nichtaxiomatisierbarkeit des unendlichwertigen Prädikatenkalküls von Łukasiewicz. Journal of Symbolic Logic 27(2), 159–170 (1962)
23. Straccia, U.: Reasoning within fuzzy description logics. Journal of Artificial Intelligence Research 14, 137–166 (2001)
24. Vojtás, P.: Fuzzy logic programming. Fuzzy Sets. and Systems 124, 361–370 (2001)

A Tableau Method for Public Announcement Logics

Philippe Balbiani[1], Hans van Ditmarsch[2],
Andreas Herzig[1], and Tiago de Lima[1]

[1] Institut de Recherche en Informatique de Toulouse, France
[2] Computer Science, University of Otago, New Zealand

Abstract. Public announcement logic is an extension of multi-agent epistemic logic with dynamic operators to model the informational consequences of announcements to the entire group of agents. We propose a labelled tableau-calculus for this logic. We also present an extension of the calculus for a logic of arbitrary announcements.

1 Introduction

Public announcement logic (PAL) was originally proposed in [1]. This is one from a series of logics developed with the aim of modelling dynamics of knowledge and belief in multi-agent settings. These logics, sometimes called dynamic epistemic logics (DELs), deal with a number of epistemic scenarios and puzzles (see [2,3,4,5] for some examples). PAL is the simplest of them. It extends epistemic logic (EL) with dynamic operators $[\varphi]$. The formula $[\varphi]\psi$ stands for 'ψ is true after the public announcement of φ'. Being the simplest form of agent communication, public announcements are present in all DELs. Some recent works, such as [6,7,8], show that they suffice to model dynamics of knowledge and belief in several cases.

Traditionally, proof systems for DELs are obtained by means of *reduction axioms*. In the particular case of PAL, they permit the translation of each PAL-formula into an equivalent EL-formula. The well-known proof system for PAL is therefore obtained by just extending that of EL by the former's reduction axioms. It follows that both logics have the same expressivity. Nevertheless the translated formula is exponentially larger than the original one. That is, PAL is strictly more succinct. This is the reason why PAL is considered to be more convenient for reasoning about knowledge [5]. Curiously however, satisfiability check in PAL is also PSPACE-complete [9].

In this paper, we present a tableau-calculus for PAL. The method decides satisfiability without reducing PAL-formulas to another language. We also extend the calculus to deal with *arbitrary public announcement logic* (APAL). It extends PAL by a modal operator $\Diamond$. The formula $\Diamond\varphi$ stands for 'there is a public announcement after which φ is true'. Note that while all other DELs deal with the question 'what becomes true after the execution of a given action?', APAL deals with the question 'is there an action whose execution makes a given formula true?'. Arbitrary public announcement logic is addressed in detail in [10].

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 43–59, 2007.

The organisation of the paper is as follows. In Section 2 we define syntax and semantics of PAL. The tableau-calculus for this logic is presented in Section 3. In Section 4 we transform it into a decision procedure. In Section 5 we expand it to a tableau-calculus for APAL. Section 6 is dedicated to some related works and discussion.

2 Syntax and Semantics of Public Announcement Logic

Assume a finite set of agents A and a countably infinite set of atoms P.

Definition 1 (Language). *The language $\mathcal{L}_{\mathrm{PAL}}$ is inductively defined by the following BNF:*

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_a\varphi \mid [\varphi]\varphi$$

where a ranges over A and p ranges over P.

For $K_a\varphi$, read 'agent a knows that φ'. For $[\varphi]\psi$, read 'after public announcement of φ, ψ is true'. Other propositional and epistemic connectives are defined by usual abbreviations. The dual of K_a is $\widehat{K}_a$ and the dual of $[\varphi]$ is $\langle\varphi\rangle$. For $\widehat{K}_a\varphi$ read 'agent a considers it possible that φ', for $\langle\varphi\rangle\psi$ read 'the announcement φ can be made and after that ψ is true'.

Definition 2 (Structures). *An* epistemic model $M = \langle W, R, V\rangle$ *consists of a non-empty set W of (factual)* states *(or 'worlds'),* accessibility $R : A \to \wp(W \times W)$, *and a* valuation $V : P \to \wp(W)$. *For $w \in W$, (M, w) is an* epistemic state *(also known as a pointed Kripke model).*

For $R(a)$ we write R_a and for $V(p)$ we write V_p. Given two states w, w' in W, the intuitive meaning of wR_aw' is that at w, the agent a considers it possible that the real world is w'.

If no restriction is imposed over the accessibility relations in R, then we call the resultant logic K-PAL. If each R_a is reflexive, then the resultant logic is called KT-PAL. If each R_a is reflexive and transitive, then we call the resultant logic S4-PAL. And finally, if each R_a is reflexive, transitive and symmetric, then we call the resultant logic S5-PAL. We continue with the semantics.

Definition 3. *Assume an epistemic model $M = \langle W, R, V\rangle$. The interpretation of an arbitrary $\varphi \in \mathcal{L}_{\mathrm{PAL}}$ is defined by induction as follows:*

$$\begin{array}{lll}
M, w \models p & \textit{iff} & w \in V_p \\
M, w \models \neg\varphi & \textit{iff} & M, w \not\models \varphi \\
M, w \models \varphi \wedge \psi & \textit{iff} & M, w \models \varphi \textit{ and } M, w \models \psi \\
M, w \models K_a\varphi & \textit{iff} & \textit{for all } v \in W,\ wR_av \textit{ implies } M, v \models \varphi \\
M, w \models [\varphi]\psi & \textit{iff} & M, w \models \varphi \textit{ implies } M|\varphi, w \models \psi
\end{array}$$

In the last clause, the epistemic model $M|\varphi = \langle W^\varphi, R^\varphi, V^\varphi\rangle$ is defined as follows:

$$\begin{array}{ll}
W^\varphi & = \{w' \in W \mid M, w' \models \varphi\} \\
R_a^\varphi & = R_a \cap\ (W^\varphi \times W^\varphi) \\
V_p^\varphi & = V_p \cap W^\varphi
\end{array}$$

Formula φ is valid in model M, notation $M \models \varphi$, if and only if for all $w \in W$, $M, w \models \varphi$. Let C be in $\{\mathrm{K}, \mathrm{KT}, \mathrm{S4}, \mathrm{S5}\}$. Formula φ is valid in C-PAL, notation $\models_{C\text{-PAL}} \varphi$, if and only if for all epistemic models $M \in C$-PAL, $M \models \varphi$.

The dynamic modal operator $[\varphi]$ is interpreted as an epistemic state transformer. Announcements are assumed to be truthful, and this is commonly known by all agents. Therefore, the model $M|\varphi$ is the model M restricted to all the states where φ is true, including access between states. For the semantics of the dual operators, we have that $M, w \models \langle\varphi\rangle\psi$ if and only if $M, w \models \varphi$ and $M|\varphi, w \models \psi$.

3 A Tableau Method for Public Announcement Logic

We present in this section a proof method for public announcement logic that uses tableaux. Exactly in the same way as all other tableau methods, given a formula φ, it systematically tries to construct a model for it. When it fails, φ is inconsistent and thus its negation is valid.

In our representation formulas are prefixed by a number that represents possible worlds in the model (similar to [11, Chapter 8]). Formulas are also prefixed by finite sequences of announcements corresponding to successive model restrictions (as in [9]). Given a finite sequence of formulas $\psi^k = (\psi_1 \ldots \psi_k)$, for each $1 \leq i \leq k$, the sequence $(\psi_1 \ldots \psi_i)$ is noted ψ^i whereas $\psi^0 = \epsilon$ denotes the empty sequence. In addition, we write $M|\psi^k$ for $M|\psi_1| \ldots |\psi_k$.

Definition 4. *A* labelled formula *is a triple $\lambda = (\psi^k, x, \varphi)$ where*

- *ψ^k is a finite sequence $(\psi_1 \ldots \psi_k)$ of formulas in $\mathcal{L}_{\mathrm{PAL}}$;*
- *$x \in \mathbb{N}$; and*
- *$\varphi \in \mathcal{L}_{\mathrm{PAL}}$.*

The part ψ^k, x is the label *of the formula φ. It represents a possible world x in the epistemic model that is successively restricted by the formulas in ψ^k.*

Definition 5. *A* skeleton *is a ternary relation $\Sigma \subseteq (A \times \mathbb{N} \times \mathbb{N})$ that represents the accessibility relations. A* branch *is a pair $b = (\Lambda, \Sigma)$ where Λ is a set of labelled formulas and Σ is a skeleton.*

Definition 6 (Tableau). *A* tableau *is a set $T^i = \{b_1^i, b_2^i, \ldots\}$ of branches. A tableau T^{i+1} is* obtained from *a tableau T^i if and only if $T^{i+1} = (T^i \setminus \{b_j^i\}) \cup B$ for some $b_j^i = (\Lambda, \Sigma) \in T^i$ and some finite set B of branches generated from b_j^i by the application of one of the* tableau rules *defined below.*

$\neg$: *if $(\psi^k, x, \neg\neg\varphi) \in \Lambda$, then $B = \{(\Lambda \cup \{(\psi^k, x, \varphi)\}, \Sigma)\}$.*
$\wedge$: *if $(\psi^k, x, \varphi_1 \wedge \varphi_2) \in \Lambda$, then $B = \{(\Lambda \cup \{(\psi^k, x, \varphi_1), (\psi^k, x, \varphi_2)\}, \Sigma)\}$.*
$\vee$: *if $(\psi^k, x, \neg(\varphi_1 \wedge \varphi_2)) \in \Lambda$, then $B = \{(\Lambda \cup \{(\psi^k, x, \neg\varphi_1)\}, \Sigma), (\Lambda \cup \{(\psi^k, x, \neg\varphi_2)\}, \Sigma)\}$.*

K: *if* $(\psi^k, x, K_a\varphi) \in \Lambda$ *and* $(a, x, x') \in \Sigma$, *then* $B = \{(\Lambda_0, \Sigma), \dots (\Lambda_k, \Sigma)\}$, *where*

$$\begin{aligned}
\Lambda_0 &= \Lambda \cup \{(\psi^0, x', \neg\psi_1)\}\\
\Lambda_1 &= \Lambda \cup \{(\psi^0, x', \psi_1), (\psi^1, x', \neg\psi_2)\}\\
\Lambda_2 &= \Lambda \cup \{(\psi^0, x', \psi_1), (\psi^1, x', \psi_2), (\psi^2, x', \neg\psi_3)\}\\
&\vdots\\
\Lambda_{k-1} &= \Lambda \cup \{(\psi^0, x', \psi_1), \dots, (\psi^{k-2}, x', \psi_{k-1}), (\psi^{k-1}, x', \neg\psi_k)\}\\
\Lambda_k &= \Lambda \cup \{(\psi^0, x', \psi_1), \dots, (\psi^{k-1}, x', \psi_k), (\psi^k, x', \varphi)\}.
\end{aligned}$$

T: *if* $(\psi^k, x, K_a\varphi) \in \Lambda$, *then* $B = \{(\Lambda \cup \{(\psi^k, x, \varphi)\}, \Sigma)\}$.

4: *if* $(\psi^k, x, K_a\varphi) \in \Lambda$ *and* $(a, x, x') \in \Sigma$, *then* $B = \{(\Lambda_1, \Sigma), \dots (\Lambda_{k+1}, \Sigma)\}$, *where*

$$\begin{aligned}
\Lambda_0 &= \Lambda \cup \{(\psi^0, x', \neg\psi_1)\}\\
\Lambda_1 &= \Lambda \cup \{(\psi^0, x', \psi_1), (\psi^1, x', \neg\psi_2)\}\\
\Lambda_2 &= \Lambda \cup \{(\psi^0, x', \psi_1), (\psi^1, x', \psi_2), (\psi^2, x', \neg\psi_3)\}\\
&\vdots\\
\Lambda_{k-1} &= \Lambda \cup \{(\psi^0, x', \psi_1), \dots, (\psi^{k-2}, x', \psi_{k-1}), (\psi^{k-1}, x', \neg\psi_k)\}\\
\Lambda_k &= \Lambda \cup \{(\psi^0, x', \psi_1), \dots, (\psi^{k-1}, x', \psi_k), (\psi^k, x', K_a\varphi)\}.
\end{aligned}$$

$\mathbf{5}_\uparrow$: *if* $(\psi^k, x, K_a\varphi) \in \Lambda$ *and* $(a, x', x) \in \Sigma$, *then* $B = \{(\Lambda_1, \Sigma), \dots (\Lambda_{k+1}, \Sigma)\}$, *where*

$$\begin{aligned}
\Lambda_0 &= \Lambda \cup \{(\psi^0, x', \neg\psi_1)\}\\
\Lambda_1 &= \Lambda \cup \{(\psi^0, x', \psi_1), (\psi^1, x', \neg\psi_2)\}\\
\Lambda_2 &= \Lambda \cup \{(\psi^0, x', \psi_1), (\psi^1, x', \psi_2), (\psi^2, x', \neg\psi_3)\}\\
&\vdots\\
\Lambda_{k-1} &= \Lambda \cup \{(\psi^0, x', \psi_1), \dots, (\psi^{k-2}, x', \psi_{k-1}), (\psi^{k-1}, x', \neg\psi_k)\}\\
\Lambda_k &= \Lambda \cup \{(\psi^0, x', \psi_1), \dots, (\psi^{k-1}, x', \psi_k), (\psi^k, x', K_a\varphi)\}.
\end{aligned}$$

$\widehat{\mathbf{K}}$: *if* $(\psi^k, x, \neg K_a\varphi) \in \Lambda$, *then* $B = \{(\Lambda \cup \{(\psi^0, x', \psi_1), \dots, (\psi^{k-1}, x', \psi_k), (\psi^k, x', \neg\varphi)\}, \Sigma \cup \{(a, x, x')\})\}$ *for some* x' *that does not appear in* Λ.

$[\cdot]$: *if* $(\psi^k, x, [\varphi_1]\varphi_2) \in \Lambda$, *then* $B = \{(\Lambda \cup \{(\psi^k, x, \neg\varphi_1)\}, \Sigma), (\Lambda \cup \{(\psi^k, x, \varphi_1), (\psi^k\varphi_1, x, \varphi_2)\}, \Sigma)\}$.

$\langle\cdot\rangle$: *if* $(\psi^k, x, \neg[\varphi_1]\varphi_2) \in \Lambda$, *then* $B = \{(\Lambda \cup \{(\psi^k, x, \varphi_1), (\psi^k\varphi_1, x, \neg\varphi_2)\}, \Sigma)\}$.

Given a formula $\varphi \in \mathcal{L}_{\mathrm{PAL}}$, *the tableau* $T^0 = \{b_1^0\} = \{(\{(\epsilon, 0, \varphi)\}, \emptyset)\}$ *is the* initial tableau *for* φ. *A* tableau for φ *is a tableau that can be obtained from the initial tableau for* φ *by successive applications of tableau rules.*

Rules $\neg$, $\wedge$ and $\vee$ are standard. The intuition behind rules $[\cdot]$ and $\langle\cdot\rangle$ reflects the semantics of public announcements. The model (M, w) satisfies $[\psi]\varphi$ if and only if (M, w) satisfies $\neg\psi$ or it satisfies ψ and the restricted model satisfies φ. Rule $\langle\cdot\rangle$ is the dual of rule $[\cdot]$. The rules for the knowledge operators are quite different from their correspondent in EL. When a new world x' is created in $M|\psi_0|\psi_1|\dots|\psi_k$ by rule $\widehat{\mathbf{K}}$, we must be sure that this world can consistently belong to M and that it is not deleted by one of the announcements of the sequence. In the case of rule **K**, the world x' was already created, but possibly in a model restricted by a different sequence of announcements. We therefore must be sure that x' would also be present in a model generated by the sequence of announcements we have in hand. This is also the case for rules **4** (transitivity) and $\mathbf{5}_{\uparrow}$ (symmetry), but not for rule **T** (reflexivity), because in the latter we do not visit a different world.

The tableau method for K-PAL consists on rules $\neg$, $\wedge$, $\vee$, $\mathbf{K}$, $\widehat{\mathbf{K}}$, $[\cdot]$ and $\langle\cdot\rangle$. For KT-PAL, we also have rule **T**. For S4-PAL, we have all rules for KT-PAL plus rule **4**. And for S5-PAL, we have all rules for S4-PAL plus rule $\mathbf{5}_{\uparrow}$.

Definition 7. *Let $b = (\Lambda, \Sigma)$ be a branch. The set Λ is* blatantly inconsistent *if and only if $\{(\psi^k, x, \varphi), (\psi^k, x, \neg\varphi)\} \subseteq \Lambda$ or $\{(\psi^k, x, p), (\chi^\ell, x, \neg p)\} \subseteq \Lambda$. The branch b is* closed *if and only if Λ is blatantly inconsistent. The branch b is* open *if and only if it is not closed. A tableau is* closed *if and only if all its branches are closed. A tableau is* open *if and only if it has at least one open branch.*

Note that (ψ^k, x, p) and $(\chi^\ell, x, \neg p)$ are inconsistent because boolean formulas are preserved through announcements.

Example 1. Consider the formula $[p\wedge\neg K_a p]\neg(p\wedge\neg K_a p)$. In Figure 1 the tableau method is used to show its validity in K-PAL. Note that in this formula the announcement corresponds to the so-called Moore sentence [6]: "p is true and agent a does not know it". When it is true and publicly announced, all the agents, in particular agent a, become aware of it. Then the sentence becomes false just after being announced. ∎

1.	$\epsilon, 0, \neg[p \wedge \neg K_a p]\neg(p \wedge \neg K_a p)$		
2.	$\epsilon, 0, p \wedge \neg K_a p$		$(\langle\cdot\rangle : 1)$
3.	$p \wedge \neg K_a p, 0, \neg\neg(p \wedge \neg K_a p)$		$(\langle\cdot\rangle : 1)$
4.	$p \wedge \neg K_a p, 0, p \wedge \neg K_a p$		$(\neg : 3)$
5.	$p \wedge \neg K_a p, 0, p$		$(\wedge : 4)$
6.	$p \wedge \neg K_a p, 0, \neg K_a p$		$(\wedge : 4)$
7.	$\epsilon, 1, p \wedge \neg K_a p$	$(a, 0, 1) \in \Sigma$	$(\widehat{\mathbf{K}} : 6)$
8.	$p \wedge \neg K_a p, 1, \neg p$		$(\widehat{\mathbf{K}} : 6)$
9.	$\epsilon, 1, p$		$(\wedge : 7)$
10.	$\epsilon, 1, \neg K_a p$		$(\wedge : 7)$
	closed		$(8, 9)$

Fig. 1. Closed tableau for the formula $[p \wedge \neg K_a p]\neg(p \wedge \neg K_a p)$

Theorem 1 (Soundness and completeness). *For $C \in \{\mathrm{K}, \mathrm{KT}, \mathrm{S4}, \mathrm{S5}\}$, there is a closed C-PAL-tableau for $\neg\varphi$ if and only if φ is C-PAL-valid.*

Proof. ($\Rightarrow$): we prove that if φ is satisfiable, then there is no closed tableau for φ. We do this by showing that all tableau rules preserve satisfiability. To do so we first need the following definition.

Definition 8. *The branch b is* satisfiable *if and only if there exists an epistemic structure $M = \langle W, R, V\rangle$ and a function f from $\mathbb{N}$ to W such that for all $(a, x, x') \in \Sigma$, $f(x)R_a f(x')$ and for all $(\psi^k, x, \varphi) \in \Lambda$:*

$$M|\psi^0, f(x) \models \psi_1,\ M|\psi^1, f(x) \models \psi_2, \dots, M|\psi^{k-1}, f(x) \models \psi_k,\ M|\psi^k, f(x) \models \varphi$$

Now, let T^i be a tableau for a given formula that contains a branch $b = (\Lambda, \Sigma)$, we show that if b is satisfiable, then the set of branches B generated by any tableau rule has also at least one satisfiable branch.

Suppose that the branch $b = (\Lambda, \Sigma)$ is satisfiable. The proofs for rules $\neg$, $\wedge$ and $\vee$ are straightforward and left to the reader. We then prove that rule **K** is sound. If $(\psi^k, x, K_a\varphi) \in \Lambda$ and $(a, x, x') \in \Sigma$, then the application of rule K generates all the branches $b_j = (\Lambda_j, \Sigma)$ for $1 \leq j \leq k+1$ such that

$$\begin{aligned}
\Lambda_1 &= \Lambda \cup \{(\psi^0, x', \neg\psi_1)\}\\
\Lambda_2 &= \Lambda \cup \{(\psi^0, x', \psi_1), (\psi^1, x', \neg\psi_2)\}\\
\Lambda_3 &= \Lambda \cup \{(\psi^0, x', \psi_1), (\psi^1, x', \psi_2), (\psi^2, x', \neg\psi_3)\}\\
&\vdots\\
\Lambda_k &= \Lambda \cup \{(\psi^0, x', \psi_1), \dots, (\psi^{k-2}, x', \psi_{k-1}), (\psi^{k-1}, x', \neg\psi_k)\}\\
\Lambda_{k+1} &= \Lambda \cup \{(\psi^0, x', \psi_1), \dots, (\psi^{k-1}, x', \psi_k), (\psi^k, x', \varphi)\}.
\end{aligned}$$

By hypothesis, there exists an epistemic structure $M = \langle W, R, V\rangle$ and a function f from $\mathbb{N}$ to W such that for all $(a, x, x') \in \Sigma$, $f(x)R_a f(x')$ and $M|\psi^0, f(x) \models \psi_1$, $\dots$, $M|\psi^{k-1}, f(x) \models \psi_k$ and $M|\psi^k, f(x) \models K_a\varphi$. Then, $M|\psi^k, f(x') \models \varphi$. Then, one of the following conditions holds:

$$\begin{aligned}
&M|\psi^0, f(x') \models \neg\psi_1 \quad \text{or}\\
&M|\psi^0, f(x') \models \varphi_1, M|\psi^1, f(x') \models \neg\psi_2 \quad \text{or}\\
&M|\psi^0, f(x') \models \varphi_1, M|\psi^1, f(x') \models \psi_2, M|\psi^2, f(x') \models \neg\psi_3 \quad \text{or}\\
&\quad\vdots\\
&M|\psi^0, f(x') \models \varphi_1, \dots, M|\psi^{k-2}, f(x') \models \psi_{k-1}, M|\psi^{k-1}, f(x') \models \neg\psi_k \quad \text{or}\\
&M|\psi^0, f(x') \models \varphi_1, \dots, M|\psi^{k-1}, f(x') \models \psi_k, M|\psi^k, f(x') \models \varphi.
\end{aligned}$$

Therefore one of the branches b_j is satisfiable.

Rules **T**, **4** and $\mathbf{5}_\uparrow$ are proved to be sound in a similar way. In these cases we also use the fact that R_a is respectively reflexive, transitive and symmetric. We omit the details here.

For rule $\widehat{\mathbf{K}}$ suppose that $(\psi^k, x, \neg K_a\varphi) \in \Lambda$. Then the application of rule $\widehat{\mathbf{K}}$ generates only one branch $b_1 = (\Lambda_1, \Sigma_1)$ such that $\Lambda_1 = \Lambda \cup \{(\psi^0, x', \psi_1), \dots, (\psi^{k-1}, x', \psi_k), (\psi^k, x', \neg\varphi)\}$ and $\Sigma_1 = \Sigma \cup \{(a, x, x')\}$ for some x' that does not occur in Λ. By hypothesis, there exists an epistemic structure $M = \langle W, R, V\rangle$ and a function f from $\mathbb{N}$ to W such that $M|\psi^0, f(x) \models \psi_1$, $\dots$, $M|\psi^{k-1}, f(x) \models \psi_k$ and $M|\psi^k, f(x) \models \neg K_a\varphi$. Then, there exists $w \in W^{\psi^k}$ such that $f(x)R_a^{\psi^k} w$ and $M|\psi^0, w \models \psi_1$, $\dots$, $M|\psi^k, w \models \psi_k$ and $M|\psi^k, w \models \neg\varphi$. We thus consider the function $f' : \mathbb{N} \to W$ such that for all x that occur in Λ, $f'(x) = f(x)$ and $f'(x') = w$. Therefore, b_1 is satisfiable.

For rule $[\cdot]$ suppose that $(\psi^k, x, [\varphi_1]\varphi_2) \in \Lambda$. Then the application of rule $[\cdot]$ generates branches $b_1 = (\Lambda \cup \{(\psi^k, x, \neg\varphi_1)\}, \Sigma)$ and $b_2 = (\Lambda \cup \{(\psi^k, x, \varphi_1), (\psi^k\varphi_1, x, \varphi_2)\}, \Sigma)$. Seeing that $M|\psi^k, f(x) \models [\varphi_1]\varphi_2$ iff either $M|\psi^k, f(x) \models \neg\varphi_1$ or $M|\psi^k, f(x) \models \varphi_1$ and $M|\psi^k|\varphi_1, f(x) \models \varphi_2$, thus b_1 is satisfiable or b_2 is satisfiable.

For rule $\langle\cdot\rangle$ suppose that $(\psi^k, x, \neg[\varphi_1]\varphi_2) \in \Lambda$. Then the application of rule $\langle\cdot\rangle$ generates only one branch $b_1 = (\Lambda \cup \{(\psi^k, x, \varphi_1), (\psi^k, \varphi_1, x, \neg\varphi_2)\}, \Sigma)$. Seeing that $M|\psi^k, f(x) \models \neg[\varphi_1]\varphi_2$ iff $M|\psi^k, f(x) \models \varphi_1$ and $M|\psi^k|\varphi_1, f(x) \models \neg\varphi_2$, thus b_1 is satisfiable.

($\Leftarrow$): we show that if a saturated tableau for a given formula φ is open, then φ is satisfiable. Suppose that T^∞ is an open saturated tableau for a S5-PAL-formula φ. Then, it contains at least one open branch $b = \langle \Lambda, \Sigma \rangle$. We use this branch to construct an epistemic structure $M = \langle W, R, V \rangle$ that satisfies φ as follows:

- $W = \{x \in \mathbb{N} \mid x \text{ occurs in } \Lambda\}$;
- R_a = reflexive, transitive and symmetric closure of $\{(x, x') \mid (a, x, x') \in \Sigma\}$; and
- $V_p = \{x \mid (\psi^k, x, p) \in \Lambda \text{ for some } \psi^k\}$.

And we also define a function $f(x) = x$ for all x occurring in Λ.

Clearly, W is a non-empty set, R_a is an equivalence relation, V_p assigns a subset of W to each proposition that appears on the tableau and if $(a, x, x') \in \Sigma$, then $f(x')R_af(x)$. Thus, we now show that for all labelled formulas $\lambda = (\psi^k, x, \varphi) \in \Lambda$, we have $\mathcal{P}(\lambda)$ defined as follows:

$$\mathcal{P}(\lambda) = \begin{cases} M|\psi^0, f(x) \models \psi_1 & \text{and} \\ \quad\vdots & \\ M|\psi^{k-1}, f(x) \models \psi_k & \text{and} \\ M|\psi^k, f(x) \models \varphi & \end{cases}$$

We do this by induction on the length of the labelled formula λ that is recursively defined as follows:

$$\begin{aligned} \text{len}(p) &= 1 \\ \text{len}(\neg\varphi) &= 1 + \text{len}(\varphi) \\ \text{len}(\varphi_1 \wedge \varphi_2) &= 1 + \text{len}(\varphi_1) + \text{len}(\varphi_2) \\ \text{len}(K_a\varphi) &= 1 + \text{len}(\varphi) \\ \text{len}([\varphi_1]\varphi_2) &= 1 + \text{len}(\varphi_1) + \text{len}(\varphi_2) \\ \text{len}(\psi^k) &= \text{len}(\psi_1) + \cdots + \text{len}(\psi_k) \\ \text{len}(\psi^k, x, \varphi) &= 1 + \text{len}(\psi^k) + \text{len}(\varphi) \end{aligned}$$

The details are left to the reader. ∎

4 Decision Procedures for Public Announcement Logics

In the way the method is defined, redundant applications of tableau rules are allowed. In particular, they can be applied indefinitely often. Therefore it may never stop. In this section we define strategies for the application of the tableau rules. They are inspired by the "tableau construction" defined in [12].

When a set of labelled formulas Λ is not saturated under one or more tableau rules, we say that $\lambda \in \Lambda$ is a *witness* to this fact if the given rule, or rules, were still not applied to λ. For convenience, we further use notation $\Lambda(x)$ for the set of *labelled formulas of* x, defined by $\{(\psi^k, \varphi) \mid (\psi^k, x, \varphi) \in \Lambda\}$. And we also use notation $\Lambda(x, a)$ for the set of *labelled formulas of agent* a *in* x, defined by $\{(\psi^k, K_a\varphi) \mid (\psi^k, x, K_a\varphi) \in \Lambda\} \cup \{(\psi^k, \neg K_a\varphi) \mid (\psi^k, x, \neg K_a\varphi) \in \Lambda\}$.

The strategy defined below is for S5-PAL. It constructs a tree of nodes s whose labels are tableau branches $L(s) = (\Lambda_s, \Sigma_s)$ generated by the application of tableau rules to their antecedents.

Strategy 1. *Let $\varphi_0 \in \mathcal{L}_{\text{PAL}}$ be given. Construct a tree as follows.*

1. *Start with a single node s_0 (the root of the tree) whose label is the initial branch for φ_0, i.e., the pair $L(s_0) = (\Lambda_{s_0}, \Sigma_{s_0})$, where $\Lambda_{s_0} = \{(\epsilon, 0, \varphi_0)\}$ and $\Sigma_{s_0} = \emptyset$.*
2. *Repeat until neither step 2(a) nor step 2(b) below applies.*
   (a) World saturation: *if s is a leaf with label $L(s)$ such that $L(s)$ is open and not saturated under rules $\neg$, $\wedge$, $\mathbf{T}$, $\langle\cdot\rangle$, $\vee$, $[\cdot]$, $\mathbf{K}$, $\mathbf{4}$ and $\mathbf{5}_\uparrow$, and $\lambda \in \Lambda_s$ is a witness to this fact, then do:*
      i. if $\lambda = (\psi^k, x, \neg\neg\varphi)$ then create a successor s' such that $\Lambda_{s'} = \Lambda_s \cup \{(\psi^k, x, \varphi)\}$ and $\Sigma_{s'} = \Sigma_s$. And then go to step 2.
      ii. if $\lambda = (\psi^k, x, \varphi_1 \wedge \varphi_2)$ then create a successor s' such that $\Lambda_{s'} = \Lambda_s \cup \{(\psi^k, x, \varphi_1), (\psi^k, x, \varphi_2)\}$ and $\Sigma_{s'} = \Sigma_s$. And then go to step 2.
      iii. if $\lambda = (\psi^k, x, K_a\varphi)$ then create a successor s' such that $\Lambda_{s'} = \Lambda_s \cup \{(\psi^k, x, \varphi)\}$ and $\Sigma_{s'} = \Sigma_s$. And then go to step 2.
      iv. if $\lambda = (\psi^k, x, \neg[\varphi_1]\varphi_2)$ then create a successor s' such that $\Lambda_{s'} = \Lambda_s \cup \{(\psi^k, x, \varphi_1), (\psi^k\varphi_1, x, \varphi_2)\}$ and $\Sigma_{s'} = \Sigma_s$. And then go to step 2.
      v. if $\lambda = (\psi^k, x, \neg(\varphi_1 \wedge \varphi_2))$ then create two successors s_1 and s_2 such that $\Lambda_{s_1} = \Lambda_s \cup \{(\psi^k, x, \neg\varphi_1)\}$ and $\Sigma_{s_1} = \Sigma_s$, and $\Lambda_{s_2} = \Lambda_s \cup \{(\psi^k, x, \neg\varphi_2)\}$ and $\Sigma_{s_2} = \Sigma_s$. And then go to step 2.
      vi. if $\lambda = (\psi^k, x, [\varphi_1]\varphi_2)$ then create two successors s_1 and s_2 such that $\Lambda_{s_1} = \Lambda_s \cup \{(\psi^k, x, \neg\varphi_1)\}$ and $\Sigma_{s_1} = \Sigma_s$, and $\Lambda_{s_2} = \Lambda_s \cup \{(\psi^k, x, \varphi_1), (\psi^k\varphi_1, x, \varphi_2)\}$ and $\Sigma_{s_2} = \Sigma_s$. And then go to step 2.
      vii. if $\lambda = (\psi^k, x, K_a\varphi)$ and $(a, x, x') \in \Sigma$, then for each $i \in \{0, 1, \ldots, k-1\}$, create a successor s_i such that $\Lambda_{s_i} = \Lambda_s \cup \{(\psi^j, x', \psi_{j+1}) \mid 0 \leq j < i\} \cup \{(\psi^i, x', \neg\psi_{i+1})\}$ and $\Sigma_{s_i} = \Sigma_s$, and also create a successor node s_k such that $\Lambda_{s_k} = \Lambda_s \cup \{(\psi^j, x', \psi_{j+1}) \mid 0 \leq j < k\} \cup \{(\psi^k, x', \varphi)\}$ and $\Sigma_{s_k} = \Sigma_s$. And then go to step 2.
      viii. if $\lambda = (\psi^k, x, K_a\varphi)$ and $(a, x, x') \in \Sigma$, then for each $i \in \{0, 1, \ldots, k-1\}$, create a successor s_i such that $\Lambda_{s_i} = \Lambda_s \cup \{(\psi^j, x', \psi_{j+1}) \mid 0 \leq j < i\} \cup \{(\psi^i, x', \neg\psi_{i+1})\}$ and $\Sigma_{s_i} = \Sigma_s$, and also create a successor node

s_k such that $\Lambda_{s_k} = \Lambda_s \cup \{(\psi^j, x', \psi_{j+1}) \mid 0 \leq j < k\} \cup \{(\psi^k, x', K_a\varphi)\}$ and $\Sigma_{s_k} = \Sigma_s$. And then go to step 2.

ix. if $\lambda = (\psi^k, x, K_a\varphi)$ and $(a, x', x) \in \Sigma$, then for each $i \in \{0, 1, \ldots, k-1\}$, create a successor s_i such that $\Lambda_{s_i} = \Lambda_s \cup \{(\psi^j, x', \psi_{j+1}) \mid 0 \leq j < i\} \cup \{(\psi^i, x', \neg\psi_{i+1})\}$ and $\Sigma_{s_i} = \Sigma_s$, and also create a successor node s_k such that $\Lambda_{s_k} = \Lambda_s \cup \{(\psi^j, x', \psi_{j+1}) \mid 0 \leq j < k\} \cup \{(\psi^k, x', K_a\varphi)\}$ and $\Sigma_{s_k} = \Sigma_s$. And then go to step 2.

(b) Create a new world: *if s is a leaf with label $L(s)$ such that $L(s)$ is open, world-saturated and not saturated under rule $\widehat{\mathbf{K}}$ and $\lambda = (\psi^k, x, \neg K_a\varphi)$ is a witness to this fact, then do steps i, ii and iii below. And then go to step 2.*

i. generate a new natural number x' that does not appear in Σ_s and create a label $L' = (\Lambda', \Sigma')$, where $\Lambda' = \{(\psi^j, x', \psi_{j+1}) \mid 0 \leq j < k\} \cup \{(\psi^k, x', \neg\varphi)\}$ and $\Sigma' = \{(a, x, x')\}$.

ii. if there is a sequence of natural numbers $y_0, y_1, \ldots, y_n$ such that $y_n = x$ and for all $0 \leq i < n$, $(a, y_i, y_{i+1}) \in \Sigma_s$ and $\Lambda_s(x, a) = \Lambda_s(y_0, a)$ and $\Lambda' \subseteq \Lambda_s(y_1)$, then create a successor s' such that $\Lambda_{s'} = \Lambda_s$ and $\Sigma_{s'} = \Sigma_s \cup \{(a, x, y_1)\}$.

iii. if step 2(b)ii does not apply, then create a successor s' such that $\Lambda_{s'} = \Lambda_s \cup \Lambda'$ and $\Sigma_{s'} = \Sigma_s \cup \Sigma'$.

3. If s is a leaf and its label $L(s)$ is open, then return `true`, else return `false`.

Simple modifications of Strategy 1 above give us strategies for the other logics we consider here. A strategy for S4-PAL can be obtained by removing step 2(a)ix. By removing steps 2(a)viii and 2(a)ix, we obtain a strategy for KT-PAL. And by removing steps 2(a)iii, 2(a)viii and 2(a)ix we obtain a strategy for K-PAL.

Note that step 2(b)ii has a *loop test*. This is crucial to guarantee that the process halts for S4-PAL and S5-PAL. Before applying rule $\widehat{\mathbf{K}}$, which means that a new "world" x' will be created, it verifies that there is no loop.

We continue by proving termination. After that we prove soundness and completeness for S5-PAL only. Proofs for the other logics are similar and left to the reader. We first need a definition and a lemma.

Definition 9. *The* set of labelled sub-formulas of φ, $\mathrm{Sub}(\varphi)$, *and* the set of labelled sub-formulas of φ and its negations, $\mathrm{Sub}^+(\varphi)$, *are recursively defined as follows:*

$$\begin{aligned}
\mathrm{Sub}(p) &= \{(\epsilon, p)\} \\
\mathrm{Sub}(\neg\varphi) &= \mathrm{Sub}(\varphi) \cup \{(\epsilon, \neg\varphi)\} \\
\mathrm{Sub}(\varphi \wedge \psi) &= \mathrm{Sub}(\varphi) \cup \mathrm{Sub}(\psi) \cup \{(\epsilon, \varphi \wedge \psi)\} \\
\mathrm{Sub}(K_a\varphi) &= \mathrm{Sub}(\varphi) \cup \{(\epsilon, K_a\varphi)\} \\
\mathrm{Sub}([\psi]\varphi) &= \mathrm{Sub}(\psi) \cup \{(\psi\chi^k, \varphi') \mid (\chi^k, \varphi') \in \mathrm{Sub}(\varphi)\} \cup \{(\epsilon, [\psi]\varphi)\}
\end{aligned}$$

$$\mathrm{Sub}^+(\varphi) = \mathrm{Sub}(\varphi) \cup \{(\psi^k, \neg\varphi') \mid (\psi^k, \varphi') \in \mathrm{Sub}(\varphi)\}$$

Lemma 1

1. *$|\mathrm{Sub}(\varphi_0)| \leq \mathrm{len}(\varphi_0)$.*
2. *For all $(\psi^k, \varphi) \in \mathrm{Sub}(\epsilon, \varphi_0)$, $k \leq \mathrm{len}(\varphi_0)$.*
3. *$|\mathrm{Sub}^+(\varphi)| \leq 2 \times \mathrm{len}(\varphi)$.*

Items 1 and 2 are proved in [9] and 3 is an obvious consequence of them.

Theorem 2. *For all $\varphi \in \mathcal{L}_{\mathrm{PAL}}$, Strategy 1 creates a finite tree for φ.*

Proof. Let a $\mathcal{L}_{\mathrm{PAL}}$-formula φ be given. Because $\mathrm{Sub}^+(\varphi)$ is finite, each step generates a finite number of immediate successors. Then, by the fact that the initial tree for φ is a single node (and, in particular, it is finite), each step of the strategy generates a finite tree.

We now show that each step is applied finitely often. Let $\mathrm{len}(\varphi) = n$. By Lemma 1, the number of labelled sub-formulas of φ and its negations is bounded by $2n$. Then after $2n$ applications of step 2(a) all the leafs of the tree are world-saturated. This means that there can be at most $2n$ applications of step 2(a) between two subsequent applications of step 2(b).

Now, note that there exists at most 2^{2n} different subsets of $\mathrm{Sub}^+(\varphi)$. This means that the loop tests can fail at most 2^{2n} times. It immediately follows that step 2(b) can be applied at most 2^{2n} times. Therefore, Strategy 1 always creates a finite tree and thus always halts. ∎

Theorem 3. *For all $\varphi_0 \in \mathcal{L}_{\mathrm{PAL}}$, φ_0 is* S5-PAL*-satisfiable if and only if Strategy1 for φ_0 returns* `true`.

Proof. ($\Rightarrow$): we show that if φ is S5-PAL-satisfiable, then the tree for φ generated by Strategy 1 will have at least one leaf whose label is an open tableau branch. We do this by showing that all steps preserve satisfiability. This proof is along the lines of the first part of the proof of Theorem 1. The only remarkable difference is the step 2(b)ii: suppose that $(\psi^k, x, \neg K_a \varphi) \in \Lambda_s$ and that the loop test succeeds. This means that there is a sequence y_0,y_1,…,y_n such that $y_n = x$ and for all $0 \leq i < n$, $(a, y_i, y_{i+1}) \in \Sigma_s$, and s has a successor s' such that $\Lambda_{s'} = \Lambda_s$ and $\Sigma_{s'} = \Sigma_s \cup \{(a, x, y_1)\}$. We then consider the (*unfolded*) tableau branch $L' = (\Lambda', \Sigma')$ such that $\Lambda' = \Lambda_{s'} \cup \{(\chi^\ell, x', \varphi') \mid (\chi^\ell, y_1, \varphi') \in \Lambda_{s'}\}$ and $\Sigma' = (\Sigma_s \setminus \{(a, x, y_1)\}) \cup \{(a, x, x'), (a, x', y_2)\}$. Clearly, L' is satisfiable if and only if $L(s')$ is satisfiable. By hypothesis, there is an epistemic structure $M = \langle W, R, V \rangle$ and a function $f : \mathbb{N} \to W$ that satisfy $L(s)$. Then there exists $w \in W^{\psi^k}$ such that $f(x) R_a^{\psi^k} w$. We thus consider the function $f' : \mathbb{N} \to W$ such that for all integer x that occur in Λ', $f'(x) = f(x)$ and $f'(x') = w$. Therefore $L(s')$ is satisfiable.

($\Leftarrow$): if Strategy 1 for φ returns `true`, then the tree for φ has a leaf s such that $L(s)$ is open and saturated. Then we use this node to construct a model $M = \langle W, R, V \rangle$ that satisfies φ as follows. W contains all x that appear in Σ_s; R is the reflexive, transitive and symmetric closure of all triples $(a, x, x') \in \Sigma_s$; and each V_p contains all x such that $(\psi^k, x, p) \in \Lambda_s$ for some ψ^k. We then proceed by induction on the length of labelled formulas where the induction hypothesis is: if $L(s)$ is an open saturated branch that contains (ψ^k, x, φ') and $\text{len}(\psi^k, x, \varphi') < n$, then $M|\psi^0, x \models \psi_1, \ldots, M|\psi^{k-1}, x \models \psi_k$, and $M|\psi^k, x \models \varphi'$. This is done along the lines of the second part of the proof of Theorem 1. The details are left to the reader. ∎

The depth of the tree created in Strategy 1 is exponential on the size of the input formula. However, S5-PAL is proven to be in PSPACE [9], which means that this algorithm is not optimal. Below, we present optimal strategies for logics K-PAL and KT-PAL.

Similarly to the "tableau construction" defined in [12], instead of labelling the nodes of the tree with entire tableau branches, in our next strategy, node labels contain formulas of only one world x. Hence, we now use pairs of the form $\lambda = (\psi^k, \varphi)$ that, for convenience, are called *labelled formulas* as well (note that x is no longer necessary). But our algorithm differs from that of [12] in a crucial point: suppose that $L(s)$ contains the formula $(\psi^k, K_a\varphi)$. When an a-successor node s' of s is created, one cannot immediately add the labelled formula (ψ^k, φ) to $L(s')$. The reason is that the world s' can have been deleted by some announcement in the sequence ψ^k (Cf. tableau rules **K**, **4** and $\mathbf{5}_{\uparrow}$). Then, in step 2(b), before adding this labelled formula to $L(s')$, the algorithm "verifies" that each ψ_i is true in s'. This is implemented with an auxiliary set $\Gamma_{s'}$.

Strategy 2. *Let $\varphi_0 \in \mathcal{L}_{\text{PAL}}$ be given. Construct a tree as follows.*

1. *Start with a single node s_0 (the root of the tree) whose label is the pair $L(s_0) = (\Lambda_{s_0}, \Gamma_{s_0})$, where $\Lambda_{s_0} = \{(\epsilon, \varphi_0)\}$ and $\Gamma_{s_0} = \emptyset$.*
2. *Repeat until neither 2(a) nor 2(b) below applies:*
   (a) Local saturation: *if s is a leaf with label $L(s)$ such that $L(s)$ is open and not saturated under rules $\neg$, $\wedge$, $\vee$, **K** and **T**, and $\lambda \in \Lambda_s$ is a witness to this fact, then do:*
      i. if $\lambda = (\psi^k, \neg\neg\varphi) \in \Lambda_s$ then create a successor s' such that $\Lambda_{s'} = \Lambda_s \cup \{(\psi^k, \varphi)\}$ and $\Gamma_{s'} = \Gamma_s$. And then go to step 2.
      ii. if $\lambda = (\psi^k, \varphi_1 \wedge \varphi_2) \in \Lambda_s$ then create successor s' such that $\Lambda_{s'} = \Lambda_s \cup \{(\psi^k, \varphi_1), (\psi^k, \varphi_2)\}$ and $\Gamma_{s'} = \Gamma_s$. And then go to step 2.
      iii. if $\lambda = (\psi^k, K_a\varphi) \in \Lambda_s$ then create successor s' such that $\Lambda_{s'} = \Lambda_s \cup \{(\psi^k, \varphi)\}$ and $\Gamma_{s'} = \Gamma_s$. And then go to step 2.
      iv. if $\lambda = (\psi^k, \neg[\varphi_1]\varphi_2) \in \Lambda_s$ then create a successor s' such that $\Lambda_{s'} = \Lambda_s \cup \{(\psi^k, \varphi_1), (\psi^k\varphi_1, \varphi_2)\}$ and $\Gamma_{s'} = \Gamma_s$. And then go to step 2.
      v. if $\lambda = (\psi^k, \neg(\varphi_1 \wedge \varphi_2)) \in \Lambda_s$ then create two successors s_1 and s_2 such that $\Lambda_{s_1} = \Lambda_s \cup \{(\psi^k, \varphi_1)\}$ and $\Gamma_{s_1} = \Gamma_s$, and $\Lambda_{s_2} = \Lambda_s \cup \{(\psi^k, \varphi_2)\}$ and $\Gamma_{s_2} = \Gamma_s$. And then go to step 2.
      vi. if $\lambda = (\psi^k, [\varphi_1]\varphi_2) \in \Lambda_s$ then create two successors s_1 and s_2 such that $\Lambda_{s_1} = \Lambda_s \cup \{(\psi^k, \neg\varphi_1)\}$ and $\Gamma_{s_1} = \Gamma_s$, and $\Lambda_{s_2} = \Lambda_s \cup \{(\psi^k, \varphi_1), (\psi^k\varphi_1, \varphi_2)\}$ and $\Gamma_{s_2} = \Gamma_s$. And then go to step 2.
      vii. if $\lambda = (\psi^k, \varphi) \in \Gamma_s$ then for each $i \in \{0, 1, \ldots, k-1\}$, create a successor s_i such that $\Lambda_{s_i} = \Lambda_s \cup \{(\psi^j, \psi_{j+1}) \mid 0 \leq j < i\} \cup \{(\psi^i, \neg\psi_{i+1})\}$ and $\Gamma_{s_i} = \Gamma_s \setminus \{\lambda\}$, and also create a successor s_k such that $\Lambda_{s_k} =$

$\Lambda_s \cup \{(\psi^j, \psi_{j+1}) \mid 0 \le j < k\} \cup \{(\psi^k, \varphi)\}$ and $\Gamma_{s_k} = \Gamma_s \setminus \{\lambda\}$. And then go to step 2.

(b) Create new worlds: *if s is a leaf with label $L(s)$ which is local saturated and not saturated under rule $\widehat{\mathbf{K}}$, then for each labelled formula of the form $\lambda = (\psi^k, \neg K_a\varphi) \in \Lambda_s$ that is witness to this, create an a-successor s' such that $\Lambda_{s'} = \{(\psi^j, \psi_{j+1}) \mid 0 \le j < k\} \cup \{(\psi^k, \neg\varphi)\}$ and $\Gamma_{s'} = \{(\chi^{k'}, \varphi') \mid \{(\chi^{k'}, K_a\varphi')\} \in \Lambda_s\})$. And then go to step 2.*

(c) Mark nodes: *if the node s with label $L(s)$ is not marked* `sat`*, then mark it* `sat` *if either:*

- *Λ_s is not local saturated and one of its successor is marked* `sat`*;*
- *Λ_s is local saturated, it is not blatantly inconsistent and Λ_s does not contain labelled formulas of the form $(\psi^k, \neg K_a\varphi)$; or*
- *Λ_s is local saturated and s has successors and all of them are marked* `sat`*.*

3. If the root of the tree is marked `sat`*, then return* `true`*, else return* `false`*.*

The strategy above can be modified for other two logics we consider here. For K-PAL we remove step 2(a)iii, and for S4-PAL, we replace step 2(b) by the following:

2(b') Create new worlds: *if s is a leaf with label $L(s)$ which is local saturated and not saturated under rule $\widehat{\mathbf{K}}$, then for each labelled formula of the form $\lambda = (\psi^k, \neg K_a\varphi) \in \Lambda_s$ that is a witness to this, then do steps i, ii and iii below. And then go to step 2.*

i. create a label $L' = (\Lambda', \Gamma')$, where $\Lambda' = \{(\psi^j, \psi_{j+1}) \mid 0 \le j < k\} \cup \{(\psi^k, \neg\varphi)\}$ and $\Gamma' = \{(\chi^{k'}, \varphi'), (\chi^{k'}, K_a\varphi') \mid (\chi^{k'}, K_a\varphi') \in \Lambda_s\})$.

ii. if there is no node s'' in the path from the root to s such that $L_{s''} = L'$, then create an a-successor node s' with label $L(s') = L'$.

iii. if step 2(b)ii does not apply, then create an a-arrow to the node s'' such that $L(s'') = L'$.

Note that we also have a loop test in step 2(b')ii. The idea is the same as in Strategy 1, but here, we also compare the sets Γ. We also remark that it is not possible to use the same idea to define a strategy for S5-PAL. We address this question in Section 6. In the sequel, we prove termination, soundness and completeness for S4-PAL only. We leave other cases to the reader.

Theorem 4. *For all $\varphi \in \mathcal{L}_{\mathrm{PAL}}$, Strategy 2 creates a finite tree for φ.*

Proof. The proof is essentially the same as for Theorem 2. ∎

Theorem 5. *For all $\varphi \in \mathcal{L}_{\mathrm{PAL}}$, φ is* S4-PAL*-satisfiable if and only if Strategy2 for φ returns* `true`*.*

Proof. From the left to the right. We show that if φ is S4-PAL-satisfiable, then the tree for φ generated by Strategy 2 has its root marked `sat`. We do this by showing that all steps preserve satisfiability. This proof is along the lines of the first part of proof of Theorem 1. The differences are steps 2(a)vii and 2(b'). Note that step 2(a)vii performs essentially the same task as steps 2(a)vii and 2(a)viii of Strategy 1. So their proof of soundness is very similar. For step 2(b'), note that it is similar to step 2(b) of Strategy 1. So its proof of soundness is also similar. We omit details here.

From the right to the left. If Strategy 2 for φ returns `true`, then the root s_0 is marked `sat`. Then, there is a sub-tree such that all its nodes are marked `sat`. We use this tree to construct a model $M = \langle W, R, V\rangle$ that satisfies φ in the following way. W contains all s in the sub-tree such that s is local saturated; each R_a is the reflexive and transitive closure of pairs (s, s') of nodes in W such that s' is a descendent of an a-successor of s; and each V_p contains all $s \in W$ such that $(\psi^k, p) \in \Lambda_s$ for some ψ^k. The proof continues along the lines of the second part of the proof of Theorem 1. We omit details here. ∎

We continue by showing computational complexity of Strategy 2 for K-PAL and KT-PAL. Remark that no optimal procedure is achieved for S4-PAL. We discuss this in Section 6.

Theorem 6 (Complexity). *The tableau system for* K-PAL *and* KT-PAL *can be implemented in polynomial space.*

Proof. We first show that the height of the trees, generated by Strategy 2, are polynomial in len(φ). The tree construction starts with the root node s_0 whose label is $L(s_0) = (\{\lambda_0\}, \emptyset)$. Suppose that len$(\lambda_0) = n$. Note that by Lemma 1 step 2(a) can be applied at most $2n$ times before generating a node s such that Λ_s is either closed or saturated. Also note that at this stage Γ_s is empty. Now, suppose that Λ_s is local saturated. Also suppose that s' is a local saturated descendent of an a-successor of s. Note that the K-modal depth of $\Lambda_{s'}$ is less than that of Λ_s. Because the number of K_a operators in φ is at most n, the root of the tree can have at most n a-descendents in the same branch of the tree. From this fact and the observation made before, it follows that the tree has height at most $\mathcal{O}(n^2)$.

We now prove that a depth first exploration of the trees can be made using a polynomial amount of memory. To see this, remember that by Lemma 1 we have that for all nodes s in the tree, $|L(s)| \leq 4n$. Then we can use a vector of $4n$ bits to encode the label of each node in the tree. We do this by setting to 1 the bit that corresponds to the formulas that are present in $L(s)$. Each step of Strategy 2 can produce at most $2n$ different immediate successors. Then, for each node, we can use a vector of $4n^2$ bits to memorise all the choices to be explored after the backtrack. It follows that we need at most $\mathcal{O}(n^5)$ bits of memory to explore the entire tree. ∎

5 A Tableau Method for Arbitrary Announcement Logic

Consider the extension of public announcement logic proposed in [10] wherein we can express what becomes true, whether known or not, without explicit reference to announcements realising that. For example, when p is true, it becomes known by announcing it:

$$\langle p\rangle K_a p$$

We can also describe ‘there is an announcement after which the agent knows p’ straightforwardly as

$$\Diamond K_a p$$

In case p is false, we can achieve $\Diamond K_a \neg p$ instead. The formula $\Diamond(K_a p \vee K_a \neg p)$ is valid. We call the logic *arbitrary public announcement logic*. For more information, see [10].

Definition 10 (Language and semantics). *The language $\mathcal{L}_{\mathrm{APAL}}$ of arbitrary public announcement logic is inductively defined as*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_a\varphi \mid [\varphi]\varphi \mid \Box\varphi$$

where $a \in A$ and $p \in P$. The extra clause needed for the semantics is as follows (note the restriction to the language of PAL*):*

$$M, w \models \Box\varphi \quad \textit{iff} \quad \textit{for all}\ \psi \in \mathcal{L}_{\mathrm{PAL}},\ M, w \models [\psi]\varphi$$

For $\Box\varphi$, read ‘after every public announcement, φ is true’. The dual of $\Box$ is $\Diamond$. For $\Diamond\varphi$, read ‘there is an announcement after which φ’. For the semantics of the dual operator, we have that: $M, w \models \Diamond\varphi$ iff there is a $\psi \in \mathcal{L}_{\mathrm{PAL}}$ such that $M, w \models \langle\psi\rangle\varphi$.

Similarly as done in Section 2, we define K-APAL, KT-APAL, S4-APAL and S5-APAL.

Example 2. A valid formula of the logic is $\Diamond(K_a p \vee K_a \neg p)$. To prove this, let (M, w) be arbitrary. Either $M, w \models p$ or $M, w \models \neg p$. In the first case, $M, w \models \Diamond(K_a p \vee K_a \neg p)$ because $M, w \models \langle p\rangle(K_a p \vee K_a \neg p)$ – the latter is true because $M, w \models p$ and $M|p, w \models K_a p$; in the second case, we analogously derive $M, w \models \Diamond(K_a p \vee K_a \neg p)$ because $M, w \models \langle\neg p\rangle(K_a p \vee K_a \neg p)$. ∎

We now provide an extension of the tableau method for public announcement logic to a tableau method for arbitrary public announcement logic. We reuse labelled formulas, skeleton and branch as introduced in Definitions 4 and 5, as well as the notions of closed and open branch as in Definition 7.

Definition 11 (Tableau (continuation)). *A* tableau *for the formula $\varphi \in \mathcal{L}_{\mathrm{APAL}}$ is defined as in Definition 6. The tableau rules are the same, plus the following ones.*

- $\Box$: *If $(\psi^k, n, \Box\varphi) \in \Lambda$, then $B = \{\langle \Lambda \cup \{(\psi^k : n : [\chi]\varphi)\}, \Sigma\rangle\}$ for any $\chi \in \mathcal{L}_{\mathrm{PAL}}$.*
- $\Diamond$: *If $(\psi^k, n, \neg\Box\varphi) \in \Lambda$, then $B = \{\langle \Lambda \cup \{(\psi^k : n : \neg[p]\varphi)\}, \Sigma\rangle\}$ for some $p \in P$ that does not occur in Λ.*

These rules are similar to Smullyan’s tableau rules for closed first-order formulas [13,14]. They reflect that the operator $\Box$ quantifies over announcements. In tableau rule $\Box$, this operator is eliminated by replacing it by an arbitrary $\mathcal{L}_{\mathrm{PAL}}$-formula. Tableau rule $\Diamond$ is more curious though: instead of replacing the operator by an announcement of a $\mathcal{L}_{\mathrm{PAL}}$-formula ψ, we replace it by an announcement of a new propositional letter. The intuitive argument here is the following. Since this new propositional letter does not occur in the branch, we are free to give it an arbitrary interpretation to represent a specific restriction in the model. In this way, we make the calculus simpler because it is not necessary to make a 'good choice' at the moment of the application of rule $\Diamond$.

Example 3. Consider the formula $[\Diamond K_a p]K_a p$. Note that it is valid in S5-APAL since its announcement corresponds to the sentence: 'there is an announcement after which agent a knows that p'. That is, it is publicly announced that p can be known. This means that p is true and thus now agent a knows it. In Figure 2 we use the tableau method to show that this formula is S5-APAL-valid. ∎

1.	$\epsilon, 0, \neg[\neg\Box\neg K_a p]K_a p$		
2.	$\epsilon, 0, \neg\Box\neg K_a p$		$(\langle\cdot\rangle : 1)$
3.	$\neg\Box\neg K_a p, 0, \neg K_a p$		$(\langle\cdot\rangle : 1)$
4.	$\epsilon, 0, \neg[p]\neg K_a p$		$(\Diamond : 2)$
5.	$\epsilon, 0, p$		$(\langle\cdot\rangle : 4)$
6.	$p, 0, \neg\neg K_a p$		$(\langle\cdot\rangle : 4)$
7.	$p, 0, K_a p$		$(\neg : 6)$
8.	$\epsilon, 1, \neg\Box\neg K_a p$	$(a, 0, 1) \in \Sigma$	$(\widehat{\mathbf{K}} : 3)$
9.	$\neg\Box\neg K_a p, 1, \neg p$		$(\widehat{\mathbf{K}} : 3)$
10.	$p, 1, p$		$(\mathbf{K} : 7)$
	closed		$(9, 10)$

Fig. 2. Closed tableau for the formula $[\Diamond K_a p]K_a p$

Theorem 7 (Soundness and completeness (continuation)). *For all $\varphi \in \mathcal{L}_{\mathrm{APAL}}$, there is a closed tableau for $\neg\varphi$ if and only if φ is* S5-APAL-*valid.*

Proof. This is an easy extension of the proof of theorem 1 (details are omitted). ∎

This tableau method can be used for giving us a proof of semi-decidability of this logic.

Theorem 8. *The set of* S5-APAL-*valid formulas of $\mathcal{L}_{\mathrm{APAL}}$ is recursively enumerable.*

Proof. First note that the same argument used in the proof of Theorem 2 can be used to show that each tableau rule generates a finite tableau. Then, by completeness, we have that for all formulas φ, all closed tableaux for φ are finite. Then, consider a procedure that enumerates all pairs (φ, T) such that T is a closed tableau. For each pair, the procedure verifies if T is a tableau for $\neg\varphi$. When the checking is finished, it generates another pair and performs another round of checking, and so on ad infinitum. ∎

6 Related Work and Discussion

We considered versions of PAL where the underlying epistemic logic obeys combination of principles T, 4 and 5. We did not consider the axiom D ($K_a\varphi \to \neg K_a\neg\varphi$) alone, i.e., epistemic logics such as KD and KD45. The reason is that in both systems the axiom T ($K_a\varphi \to \varphi$) is derivable for any boolean formula φ. To see this, note that if we have axiom D, then $(K_a\varphi \wedge \neg\varphi) \to \langle\neg\varphi\rangle\bot$ is valid for any boolean formula φ.

Recently, another tableau method for S5-PAL was proposed by [15]. Apart from some aesthetic differences, this method is very similar to ours. However, no proof of decidability is provided.

Our Strategy 2 is based on the optimal strategy for EL presented in [12]. Note however that instead of our rule $\mathbf{5}_\uparrow$, Halpern and Moses use a rule that propagates all formulas prefixed by K_a and $\widehat{K}_a$ operators to the a-successors. As this rule alone is not complete for S5, they also need to saturate the nodes under sub-formulas (which is called full propositional tableau). But note that such a rule would not be sound in our setting. For example, suppose that in node s with label $L(s)$ we have that $(\psi, \neg K_a\varphi) \in \Lambda_s$. Because it may be the case that Λ_s also contains $(\epsilon, \neg\psi)$, we cannot add neither (ψ, φ) nor $(\psi, \neg\varphi)$ to Λ_s at the risk of making it blatantly inconsistent. Then, we cannot have our set of formulas saturated under sub-formulas in this way. An optimal strategy for S4-PAL seems to be impossible too. An example is the formula $\neg K_a p_0 \wedge K_a[q_1]K_a p_1 \wedge K_a[q_2]K_a p_2 \wedge \cdots \wedge K_a[q_i]K_a p_i$, for which Strategy 2 generates a tree containing a branch with 2^i different a-successors.

7 Conclusion

We provided a proof system for PAL with and without positive introspection that avoids translation to other languages as done by [9]. It also extends to APAL.

As mentioned before, proof systems for DELs are usually built from reduction axioms. In all cases, the dynamic operators can be eliminated and the formula is translated into a simpler language. However, the price is an exponentially larger formula to be evaluated. As proved here and also in [9], there are cases where this price is not mandatory. Therefore, one of the raised questions is whether direct methods, like the method presented here, can be also applied to the other languages. In this direction, we intend to consider some “natural” extensions to our approach. For instance, PAL with common knowledge.

Acknowledgements

Hans van Ditmarsch appreciates support from the NIAS (Netherlands Institute for Advanced Study in the Humanities and Social Sciences) project ‘Games, Action, and Social Software’ and the NWO (Netherlands Organisation for Scientific Research) Cognition Program for the Advanced Studies grant NWO 051-04-120.

Tiago de Lima is supported by the Programme Alßan, the European Union Programme of High Level Scholarships for Latin America, scholarship number E04D041703BR.

All the authors thank the anonymous reviewers for their useful comments.

References

1. Plaza, J.: Logics of public communications. In: Emrich, M.L., Hadzikadic, M., Pfeifer, M.S., Ras, Z.W. (eds.) Proceedings of ISMIS 1989, pp. 201–216 (1989)
2. Baltag, A., Moss, L., Solecki, S.: The logic of public announcements, common knowledge, and private suspicions. Technical Report SEN-R9922, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands (1999)
3. Gerbrandy, J.: Bisimulations on Planet Kripke. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands (1999)
4. Kooi, B.: Expressivity and completeness for public update logic via reduction axioms. Journal of Applied Non-Classical Logics (to appear, 2007)
5. van Benthem, J., van Eijck, J., Kooi, B.: Logics of communication and change. Information and Computation 204(11), 1620–1662 (2006)
6. van Ditmarsch, H., Kooi, B.: The secret of my success. Synthese 151, 201–232 (2006)
7. van Ditmarsch, H., van der Hoek, W., Kooi, B.: Playing cards with Hintikka. Australasian Journal of Logic 3, 108–134 (2005)
8. Herzig, A., De Lima, T.: Epistemic actions and ontic actions: A unified logical framework. In: Sichman, J.S., Coelho, H., Rezende, S.O. (eds.) IBERAMIA 2006 and SBIA 2006. LNCS (LNAI), vol. 4140, pp. 409–418. Springer, Heidelberg (2006)
9. Lutz, C.: Complexity and succintness of public announcement logic. In: Stone, P., Weiss, G. (eds.) Proceedings of AAMAS, 137–144 (2006)
10. Balbiani, P., Baltag, A., van Ditmarsch, H., Herzig, A., Hoshi, T., de Lima, T.: What can we achieve by arbitrary announcements? A dynamic take on Fitch's knowability. In: Proceedings of TARK (to appear, 2007)
11. Fitting, M.: Proof Methods for Modal and Intuitionistic Logics. Reidel Publishing Company (1983)
12. Halpern, J., Moses, Y.: A guide to completeness and complexity for modal logics of knowledge and belief. Artificial Intelligence 54, 311–379 (1992)
13. Smullyan, R.M.: First-Order Logic. Springer-Verlag (1968)
14. Letz, R.: Tableau methods for modal and temporal logics. In: D'Agostino, M., Gabbay, D., Hähnle, R., Possega, J. (eds.) Handbook of Tableau Methods, Kluwer Academic Publishers, Boston (1999)
15. de Boer, M.: Praktische bewijzen in public announcement logica (Practical proofs in public announcement logic). Master's thesis, Department of Artificial Intelligence, University of Groningen, in Dutch (2006)

Bounded Model Checking with Description Logic Reasoning

Shoham Ben-David, Richard Trefler, and Grant Weddell

David R. Cheriton School of Computer Science
University of Waterloo

Abstract. Model checking is a technique for verifying that a finite-state concurrent system is correct with respect to its specification. In *bounded* model checking (BMC), the system is unfolded until a given depth, and translated into a CNF formula. A SAT solver is then applied to the CNF formula, to find a satisfying assignment. Such a satisfying assignment, if found, demonstrates an error in the model of the concurrent system.

Description Logic (DL) is a family of knowledge representation formalisms, for which reasoning is based on tableaux techniques. We show how Description Logic can serve as a natural setting for representing and solving a BMC problem. We formulate a bounded model checking problem as a consistency problem in the DL dialect $\mathcal{ALCI}$. Our formulation results in a compact representation of the model, one that is linear in the size of the model description, and does not involve any unfolding of the model. Experimental results, using the DL reasoner FaCT^{++}, significantly improve on a previous approach that used DL reasoning for model checking.

1 Introduction

Model checking ([7,20], c.f.[8]) is a technique for verifying finite-state concurrent systems, that has been proven to be very effective in the verification of hardware and software programs. In model checking, a model M, given as a set of state variables V and their next-state relations, is verified against a temporal logic formula φ. Essentially, verification of the formula φ on a model M, checks that the tree of all computations of M satisfies φ.

The main challenge in model checking is known as the *state space explosion* problem, where the number of states in the model grows exponentially in the number of variables describing it. To cope with this problem, model checking is done *symbolically*, by representing the system under verification as sets of states and transitions, and by using Boolean functions to manipulate those sets. Two main symbolic methods are used to perform model checking. The first, known as *SMV* [16], is based on Binary Decision Diagrams (BDDs) [5] for representing the state space as well as for performing the model checking procedure. The second is known as Bounded Model Checking (*BMC*) [4]. Using this method, the model under verification is unfolded k times (for a given bound k), and translated into a propositional CNF formula. A SAT solver is then applied to the formula, to find a satisfying assignment. Such an assignment, if found, demonstrates an error in the model.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 60–72, 2007.

Description Logic (DL) ([1]) is a family of knowledge representation formalisms mainly used for specifying ontologies for information systems. An ontology $\mathcal{T}$ is called a *terminology* or more simply a *Tbox*, and corresponds to a set of concept inclusion dependencies. Each inclusion dependency has the form $\mathsf{C}_1 \sqsubseteq \mathsf{C}_2$, and asserts containment properties of relevant concepts in an underlying domain, e.g., that *managers* are included in *employees*

$$\text{MANAGER} \sqsubseteq \text{EMPOYEE},$$

and also in *those things that hire only employees*

$$\text{MANAGER} \sqsubseteq \forall \mathit{hires}.\text{EMPLOYEE}.$$

In this latter case, *hires* is an example of a *role*. In DLs, a role is always a binary relation over the underlying domain.

The main reasoning service provided by a DL system is *concept consistency*, that is, for a given terminology $\mathcal{T}$ and concept C, to determine if there is a non-empty interpretation of the concept that also satisfies each inclusion dependency in $\mathcal{T}$, written $\mathcal{T} \models_{dl} \mathsf{C}$. Most DL systems implement this service by employing some form of tableaux or model building techniques. The examples illustrate that these techniques manifest both propositional and modal reasoning (where *hires* is viewed as an event), which makes using a DL system an attractive possibility for model checking.

To explore this, we consider a goal directed embedding of BMC problems as concept consistency problems in the DL dialect $\mathcal{ALCI}$. Our encoding of a model description as a terminology in $\mathcal{ALCI}$ results in a natural *symbolic* representation of the sets of states and state transitions. Specifically, given a model description MD and a bound k, we formulate a *BMC* problem as a terminology $\mathcal{T}^k_{MD}$ over $\mathcal{ALCI}$. Our formulation is compact, and does not involve *unfolding* of the model. Rather, the size of the terminology is the same as the size of the description of the model plus a set of k concept inclusions that are needed for the bounded verification. In contrast, the known *BMC* method that uses a SAT solver for this task needs k copies of the model description. This produces a representation that is k times larger than ours. For simplicity, we assume the formula to be verified expresses a *safety* property (an $AG(b)$ type formula). We note that a large and useful subset of temporal logic formulas can be translated into $AG(b)$ type formulas [2].

Let M be a model defined by a set V of Boolean state variables and their next-state transitions R. We represent each variable $v_i \in V$ as a concept V_i and the transition relation as a single role R. We then introduce concept inclusions of the type

$$\mathsf{C}_1 \sqsubseteq \forall \mathsf{R}.\mathsf{C}_2$$

stating that if the current state satisfies the condition represented by C_1, then all the next-states that can be reached in one step by R, must satisfy the condition C_2. Note that interpretations for this set of concept inclusions correspond to sub-models of the given model M.

Let the concept S_0 represent the set of initial states of M. If S_1 represents states that can be reached in one step from S_0, then the concept inclusion $\mathsf{S}_1 \sqsubseteq \exists \mathsf{R}^-.\mathsf{S}_0$ must hold (that is, the set S_1 is a subset of all the states that can reach S_0 by going one step backwards using the relation R). Similarly, we denote by S_i subsets of the states reachable in i steps from the set of initial states, and introduce the inclusions

$$\mathsf{S}_i \sqsubseteq \exists \mathsf{R}^-.\mathsf{S}_{i-1}$$

for $0 < i \leq k$. Let $\varphi = AG(b)$ be the specification to be verified, and let B be the concept representing b (composed of a Boolean combination of the concepts V representing the state variables). Model checking is now carried out by asking the query: "does there exist an interpretation for the above set of concept inclusions, such that $\mathsf{C}_\varphi (= \neg\mathsf{B} \sqcap \mathsf{S}_i)$ is not empty for some S_i?". A positive answer from the DL reasoner indicates an error in M.

We relate the consistency of the concept C_φ with respect to the terminology $\mathcal{T}_M^k$ to the satisfaction of φ in the model M, by proving that $M^k \not\models \varphi$ if and only if $\mathcal{T}_M^k \models_{dl} \mathsf{C}_\varphi$ is consistent.

Note that this formulation of a model checking problem is *goal directed*. That is, the DL reasoner begins from a description of buggy states ($\neg\mathsf{B} \sqcap \mathsf{S}_i$), and proceeds from there to find a legal backward path to a description of initial states. In earlier preliminary work using a DL for model checking [4], we explored a synchronous forward reasoning approach. In comparison to this earlier approach, our experimental results confirm that goal directed encodings perform far better, indeed outperforming a BDD-based technology for the sample safety property considered. However, the combination of our current encoding and current DL reasoning technology [14] is still not competitive with SAT-based approaches. We give some suggestions for future work to address this in our concluding remarks.

The rest of the paper is organized as follows. The next section provides the necessary background definitions. Our main contributions then follow in Section 3 in which we formally define our translation and prove its correctness, and we report on some preliminary experimental results. In Section 4 we discuss related work. Summary comments and conclusions then follow in Section 5.

2 Background and Definitions

2.1 Description Logic

Description Logics [1] come in different dialects. The basic DL dialect is called *Attributive Language* with *Complements*, or $\mathcal{ALC}$. For our purposes we need the more expressive dialect $\mathcal{ALCI}$, allowing the use of role inverse. Its definition is given below.

Definition 1 (Description Logic $\mathcal{ALCI}$). *Let* NC *and* NR *be sets of atomic concepts* $\{A_1, A_2, \ldots\}$ *and atomic roles* $\{R_1, R_2, \ldots\}$ *respectively. The set of* roles R *of the description logic* $\mathcal{ALCI}$ *is the smallest set including* NR *that satisfies the following.*

- *If* $R \in \mathsf{R}$ *then so is* R^-.

The set of concepts C *of the description logic* $\mathcal{ALCI}$ *is the smallest set including* NC *that satisfies the following conditions.*

- *If* $C_1, C_2 \in \mathsf{C}$ *then so are* $\neg C_1$ *and* $C_1 \sqcap C_2$.
- *If* $C \in \mathsf{C}$ *and* $R \in \mathsf{R}$ *then* $\exists R.C \in \mathsf{C}$.

Additional concepts are defined as syntactic sugaring of those above:

- $\top = A \sqcup \neg A$ for some A
- $\forall R.C = \neg \exists R.\neg C$
- $C_1 \sqcup C_2 = \neg(\neg C_1 \sqcap \neg C_2)$

An *inclusion dependency* is an expression of the form $C_1 \sqsubseteq C_2$. A *terminology* $\mathcal{T}$ consists of a finite set of inclusion dependencies.

The *semantics* of expressions is defined with respect to a structure $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set, and $\cdot^{\mathcal{I}}$ is a function mapping every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that the following conditions are satisfied.

- $(R^-)^{\mathcal{I}} = \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (y, x) \in R^{\mathcal{I}} \}$
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
- $(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \sqcap C_2^{\mathcal{I}}$
- $\exists R.C = \{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}}$ s.t. $(x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$

A structure *satisfies an inclusion dependency* $C_1 \sqsubseteq C_2$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$. The *consistency problem for* $\mathcal{ALCI}$ asks if $\mathcal{T} \models_{dl} C$ holds[1], that is, if there exists $\mathcal{I}$ such that $C^{\mathcal{I}}$ is non-empty and such that $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ holds for each $C_1 \sqsubseteq C_2$ in $\mathcal{T}$.

2.2 Symbolic Model Checking

Definition 2 (Kripke Structure). *Let V be a set of Boolean variables. A* Kripke structure *M over V is a four tuple $M = (S, I, R, L)$, where*

1. *S is a finite set of states.*
2. *$I \subseteq S$ is the set of initial states.*
3. *$R \subseteq S \times S$ is a transition relation that must be total, that is, for every state $s \in S$ there is a state $s' \in S$ such that $R(s, s')$.*
4. *$L : S \to 2^V$ is a function that labels each state with the set of variables true in that state.*

We view each state s as a truth assignment to the variables V. We view a set of states as a Boolean function over V characterizing the set. Thus, if a state s belongs to a set S_0, we write $s \models S_0$. Similarly, if $v_i \in L(s)$ we write $s \models v_i$, and if $v_i \notin L(s)$ we write $s \models \neg v_i$. We say that $w = s_0, s_1, ..., s_k$ is a path in M if $\forall i$, $0 \leq i < k$, $(s_i, s_{i+1}) \in R$ and $s_0 \models I$.

In practice, the full Kripke structure of a system is not explicitly given. Rather, a model is given as a set of Boolean variables $V = \{v_1, ..., v_n\}$, their initial values and their next-state assignments. The definition we give below is an abstraction of the input language of *SMV* [16].

Definition 3 (Model Description). *Let $V = \{v_1, ..., v_n\}$ be a set of Boolean variables. A tuple $MD = (I_{MD}, [\langle c_1, c_1' \rangle, ..., \langle c_n, c_n' \rangle])$ is a* Model Description *over V where I_{MD}, c_i, c_i' are Boolean expressions over V.*

[1] We write "$\models_{dl}$" to distinguish the use of the double turnstyle symbol by both description logic and model checking communities.

The semantics of a model description is a Kripke structure $M_{MD} = (S, I_M, R, L)$, where $S = 2^V$, $L(s) = s$, $I_M = \{s | s \models I_{MD}\}$, and $R = \{(s, s') : \forall 1 \leq i \leq n,\ s \models c_i$ implies $s' \models \neg v_i$ and $s \models c'_i \wedge \neg c_i$ implies $s' \models v_i\}$.

Intuitively, a pair $\langle c_i, c'_i \rangle$ defines the next-state assignment of variable v_i in terms of the current values of $\{v_1, ..., v_n\}$. That is,

$$\text{next}(v_i) = \begin{cases} 0 & \text{if } c_i \\ 1 & \text{if } c'_i \wedge \neg c_i \\ \{0,1\} & \text{otherwise} \end{cases}$$

where the assignment $\{0, 1\}$ indicates that for every possible next-state value of variables $v_1, ...v_{i-1}, v_{i+1}, ..., v_n$ there must exist a next-state with $v_i = 1$, and a next-state with $v_i = 0$.

Safety Formulas. The formulas we consider are *safety* formulas, given as $AG(b)$ in CTL [7], or $G(b)$ in LTL [19]. Such formulas state that the Boolean expression b holds on all reachable states of the model under verification. We note that a large and useful subset of CTL and LTL can be translated into $AG(b)$ type formulas [2].

Bounded Model Checking. Given a Kripke structure M, a formula φ, and a bound k, Bounded Model Checking (BMC) tries to refute $M \models \varphi$ by proving the existence of a witness to the negation of φ, of length k or less. For $\varphi = AG(b)$ we say that $M^k \not\models \varphi$ if and only if there exists a path $w = s_0, ..., s_j$, such that $j \leq k$ and $s_j \models \neg b$.

The original BMC method [4] generates a propositional formula that is satisfiable if and only if $M^k \not\models \varphi$. We show how to achieve this using Description Logic.

3 Bounded Model Checking Using Description Logic

We give a linear reduction of a bounded model checking problem into a consistency check over $\mathcal{ALCI}$. Our method performs bounded reachability on the given model, and thus resembles the BMC [4] method. However, classical BMC methods unfold the model k times (for a bound k), introducing k copies of the state variables, as well as of the transition relation. Our method in contrast, uses only *one* copy of each state variable, and defines reachability of bound k as a set of k concept inclusions. Thus our method resembles the reachability algorithm performed in BDD-based symbolic model checking [16]. Our method can therefore be seen as a combination of the two major approaches currently existing for symbolic model checking.

In the next Section we present the translation into a DL terminology. We demonstrate the translation using an example in section 3.2, and then prove the correctness of the translation in Section 3.3. In Section 3.4 we discuss implementation and experimental results.

3.1 Constructing a Terminology over $\mathcal{ALCI}$

Let $MD = (I, [\langle c_1, c'_1 \rangle, ..., \langle c_n, c'_n \rangle])$ be a model description for the model $M_{MD} = (S, I, R, L)$, over $V = \{v_1, ..., v_n\}$. Let k be the bound and let φ be a safety formula.

We generate a terminology $\mathcal{T}_{MD}^{k}$, linear in the size of $MD+k$, and a concept C_{φ}, linear in the size of φ, such that $\mathcal{T}_{MD}^{k} \models C_{\varphi}$ is consistent if and only if $M_{MD}^{k} \not\models \varphi$.

For each variable $v_i \in V$ we introduce one primitive concept V_i, where V_i denotes $v_i = 1$ and $\neg\mathsf{V}_i$ denotes $v_i = 0$. We introduce one primitive role R corresponding to the transition relation of the model.

We construct the terminology $\mathcal{T}_{MD}^{k}$ as the union of two terminologies: $\mathcal{T}_{MD}^{k} = \mathcal{T}_{MD} \cup \mathcal{T}_{k}$, where the terminology $\mathcal{T}_{MD}$ depends on the model description, and $\mathcal{T}_{k}$ depends only on the bound k on the number of cycles searched. The construction of $\mathcal{T}_{MD}$ and $\mathcal{T}_{k}$ are given below.

Constructing $\mathcal{T}_{MD}$. Let $MD = (I, [\langle c_1, c_1' \rangle, ..., \langle c_n, c_n' \rangle])$ be a model description, where I, c_i, c_i' are Boolean expressions over the variables $V = \{v_1, ..., v_n\}$. We define the concept S_0 to represent I, by replacing each v_i in I with the concept V_i, and the connectives $\wedge, \vee, \neg$ with $\sqcap, \sqcup, \neg$.

Let the pair $\langle c_i, c_i' \rangle$ describe the next state behavior of the variable v_i, as shown in Definition 3. Let C_i be the concept generated by replacing every v_i in c_i with the concept V_i, and the connectives $\wedge, \vee, \neg$ with $\sqcap, \sqcup, \neg$. Let C_i' be the concept corresponding to c_i' in the same way. We introduce the following concept inclusions.

$$\begin{aligned} \mathsf{C}_i &\sqsubseteq \forall \mathsf{R}.\neg\mathsf{V}_i \\ (\neg\mathsf{C}_i \sqcap \mathsf{C}_i') &\sqsubseteq \forall \mathsf{R}.\mathsf{V}_i \end{aligned}$$

In total, two concept inclusions are introduced for each variable v_i in MD (corresponding to the pair $\langle c_i, c_i' \rangle$).

Constructing $\mathcal{T}_{k}$. For a bound k, we introduce k primitive concepts, $\mathsf{S}_1, ..., \mathsf{S}_k$. For $1 \leq i \leq k$, we introduce k inclusions:

$$\mathsf{S}_i \sqsubseteq \exists \mathsf{R}^{-}.\mathsf{S}_{i-1},$$

Note that the concept inclusions in $\mathcal{T}_{k}$ are purely syntactic and do not depend on the model description under verification MD. In fact, the same set of inclusions shall appear in the verification (of bound k) of any model.

Constructing C_{φ}. Let φ be the specification to be verified. As mentioned before, we are concerned with safety formulas, asserting "$AG(b)$", with b being a Boolean formula over the variables $v_1, ..., v_n$. To show that such a formula does not hold, it is enough to find one state s of the Kripke structure, reachable from the initial state, such that $s \models \neg b$. We translate the Boolean formula b into a concept B in the usual way, where each variable v_i is translated to the concept V_i, and the Boolean connectives $\vee, \wedge, \neg$ into their correspondents $\sqcup, \sqcap, \neg$.

We define the concept $\mathsf{C}_{\varphi} \equiv \neg\mathsf{B} \sqcap (\mathsf{S}_0 \sqcup \mathsf{S}_1 \sqcup ... \sqcup \mathsf{S}_k)$. If C_{φ} is consistent with respect to the terminology $\mathcal{T}_{MD}^{k} = \mathcal{T}_{k} \cup \mathcal{T}_{MD}$ it means that $\neg b$ holds in some state, with distance less than k from the initial state. Verification is therefore reduced to the query: $\mathcal{T}_{MD}^{k} \models_{dl} \mathsf{C}_{\varphi}$.

3.2 Example

Consider the model description

$$\mathtt{Exmp} = (I, [\langle v_1 \wedge v_2, v_3\rangle, \langle \neg v_2, v_1 \wedge \neg v_1\rangle, \langle \neg v_1, v_1\rangle])$$

over $V = \{v_1, v_2, v_3\}$ with $I = \neg v_1 \wedge v_2 \wedge \neg v_3$. Figure 1 draws the states and transitions of the Kripke structure $M_{\mathtt{Exmp}}$ described by $\mathtt{Exmp}$, where the label of each state is the value of the vector (v_1, v_2, v_3). Let the formula to be verified be $\varphi = AG(\neg v_2 \vee \neg v_3)$. Note that $M_{\mathtt{Exmp}} \not\models \varphi$, as can be seen in Figure 1, since the state $(0, 1, 1)$, that contradicts φ, can be reached in two steps from the initial state. We choose the bound to be $k = 4$.

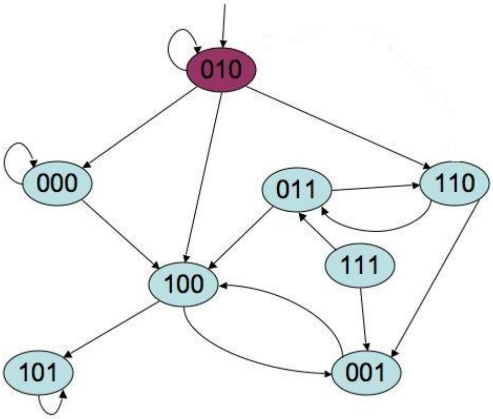


Fig. 1. A Kripke structure for Exmp

In order to build a terminology for $\mathtt{Exmp}$ we introduce one primitive role R and three primitive concepts $\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3$. We first build the terminology $\mathcal{T}_{\mathtt{Exmp}}$. For the initial state, represented by the concept S_0, we introduce the following concept inclusion:

$$\mathsf{S}_0 \sqsubseteq (\neg\mathsf{V}_1 \sqcap \mathsf{V}_2 \sqcap \neg\mathsf{V}_3)$$

The rest of $\mathcal{T}_{\mathtt{Exmp}}$ is composed of the transition relation of the model, as given below.

$$\begin{aligned}(\mathsf{V}_1 \sqcap \mathsf{V}_2) &\sqsubseteq \forall\mathsf{R}.\neg\mathsf{V}_1\\ (\neg(\mathsf{V}_1 \sqcap \mathsf{V}_2) \sqcap \mathsf{V}_3) &\sqsubseteq \forall\mathsf{R}.\mathsf{V}_1\\ \neg\mathsf{V}_2 &\sqsubseteq \forall\mathsf{R}.\neg\mathsf{V}_2\\ \neg\mathsf{V}_1 &\sqsubseteq \forall\mathsf{R}.\neg\mathsf{V}_3\\ \mathsf{V}_1 &\sqsubseteq \forall\mathsf{R}.\mathsf{V}_3\end{aligned}$$

Note that for simplicity, we omitted the inclusion $(\neg\neg\mathsf{V}_2 \sqcap \mathsf{V}_1 \sqcap \neg\mathsf{V}_1) \sqsubseteq \forall\mathsf{R}.\mathsf{V}_2$ (corresponding to $\neg\mathsf{C}_i \sqcap \mathsf{C}'_i \sqsubseteq \forall\mathsf{R}.\mathsf{V}_i$ for $i = 2$), since the prefix $\neg\neg\mathsf{V}_2 \sqcap \mathsf{V}_1 \sqcap \neg\mathsf{V}_1$ is actually equivalent to $\bot$. Similarly, the concept $\neg\neg\mathsf{V}_1 \sqcap \mathsf{V}_1$ (corresponding to $\neg\mathsf{C}_3 \sqcap \mathsf{C}'_3$) was replaced by the equivalent V_1.

In order to "unfold" the model four times (for the chosen bound $k = 4$), we introduce the primitive concepts $\mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3, \mathsf{S}_4$, and the concept inclusions:

$$\begin{array}{rcl} \mathsf{S}_1 & \sqsubseteq & \exists \mathsf{R}^-.\mathsf{S}_0 \\ \mathsf{S}_2 & \sqsubseteq & \exists \mathsf{R}^-.\mathsf{S}_1 \\ \mathsf{S}_3 & \sqsubseteq & \exists \mathsf{R}^-.\mathsf{S}_2 \\ \mathsf{S}_4 & \sqsubseteq & \exists \mathsf{R}^-.\mathsf{S}_3 \end{array}$$

For the specification $\varphi = AG(\neg v_2 \vee \neg v_3)$ we get $\mathsf{B} \equiv \neg \mathsf{V}_2 \sqcup \neg \mathsf{V}_3$, and $\mathsf{C}_\varphi \equiv \neg \mathsf{B} \sqcap (\mathsf{S}_0 \sqcup \mathsf{S}_1 \sqcup \mathsf{S}_2 \sqcup \mathsf{S}_3 \sqcup \mathsf{S}_4)$. Verification is then carried out by asking the query: Is the concept C_φ consistent with respect to $\mathcal{T}^4_{\mathtt{Exmp}}$?

In the next section we prove the correctness of our translation.

3.3 Correctness

We relate the consistency of the concept C_φ with respect to $\mathcal{T}^k_{MD}$ to the satisfaction of φ in the model M_{MD}. Let $MD = (I, [\langle c_1, c'_1 \rangle, ..., \langle c_n, c'_n \rangle])$ denote a model description for a model $M_{MD} = (S, I_M, R, L)$, and let $\varphi = AG(b)$ be a safety formula. Let $\mathcal{T}^k_{MD}$ be the terminology built for MD, as defined in Section 3.1, and let C_φ be the concept representing φ.

For the proof of the theorem, we need the following definition and lemma. Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be an interpretation for $\mathcal{T}^k_{MD}$. We define a function from the elements of $\mathcal{I}$ to states in S in the following way.

Definition 4. *F from $\Delta^\mathcal{I}$ to S is a function such that $F(\sigma) = s$ if $\forall 1 \leq i \leq n$, $\sigma \in \mathsf{V}_i$ if and only if $s \models v_i$.*

Note that the function F is well defined, since a state s is determined by the value of the variables $v_1, ..., v_n$.

Lemma 5. *Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be an interpretation for $\mathcal{T}^k_{MD}$. Let c be a Boolean expression over $v_1, ..., v_n$, and C its corresponding concept derived by replacing each variable v_i by the concept V_i, and the Boolean connectives $\vee, \wedge, \neg$ by $\sqcup, \sqcap, \neg$. Let $\sigma \in \Delta^\mathcal{I}$ be an element in the interpretation $\mathcal{I}$, and let $s = F(\sigma)$. Then $\sigma \in C^\mathcal{I}$ if and only if $s \models c$.*

Proof. By induction on the structure of the Boolean expression c. □

Theorem 6. *$M^k_{MD} \not\models \varphi$ if and only if $\mathcal{T}^k_{MD} \models_{dl} \mathsf{C}_\varphi$ is consistent.*

Proof. ($\Longrightarrow$) Assume that $M^k_{MD} \not\models \varphi$. Then there exists a path in M^k_{MD}, $w = s_0, ..., s_j$, where $j \leq k$, such that $s_0 \models I$, $\forall 0 < l \leq j, (s_{l-1}, s_l) \in R$, and $s_j \models \neg b$. We build a finite interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ based on w. The set $\Delta^\mathcal{I}$ includes $j + 1$ elements $\sigma_0, ..., \sigma_j$. Each of the primitive concepts V_i is interpreted as a set $\mathsf{V}_i^\mathcal{I}$, such that $\forall 0 \leq l \leq j$, $\sigma_l \in \mathsf{V}_i^\mathcal{I}$ if and only if $s_l \models v_i$. Note that for this interpretation, $F(\sigma_l) = s_l$.

We interpret each primitive concept S_l as $\{\sigma_l\}$ for $0 \leq l \leq j$. The primitive concepts $\mathsf{S}_{j+1}, ..., \mathsf{S}_k$ are interpreted as $\emptyset$. The interpretation $\mathsf{R}^\mathcal{I}$ of the role R is a set of pairs (σ_l, σ_{l+1}), $0 \leq l < j$. It remains to show that all concept inclusions of $\mathcal{T}^k_{MD}$ hold under this interpretation, and that $\mathsf{C}_\varphi^\mathcal{I}$, the interpretation of the concept C_φ is not empty.

- Inclusions from $\mathcal{T}_k$: For $l > j$, $\mathsf{S}_l^{\mathcal{I}} = \emptyset$, and are thus included in any other set. In order for $\mathsf{S}_l \sqsubseteq \exists \mathsf{R}^{-}.\mathsf{S}_{l-1}$ to hold, for $l \leq j$, we need to show that $\mathsf{S}_l^{\mathcal{I}} \subseteq \{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}}$ s.t. $(y,x) \in \mathsf{R}^{\mathcal{I}} \wedge y \in \mathsf{S}_{l-1}^{\mathcal{I}}\}$. Indeed, $\mathsf{S}_l^{\mathcal{I}} = \{\sigma_l\}, \mathsf{S}_{l-1}^{\mathcal{I}} = \{\sigma_{l-1}\}, (\sigma_{l-1}, \sigma_l) \in \mathsf{R}^{\mathcal{I}}$, and (σ_{l-1}, σ_l) is the only pair $(x,y) \in \mathsf{R}^{\mathcal{I}}$ such that $x \in \mathsf{S}_{l-1}^{\mathcal{I}}$. Thus the inclusion holds.
- Inclusions from $\mathcal{T}_{MD}$: We need to show that inclusions of the type $\mathsf{C}_i \sqsubseteq \forall \mathsf{R}.\neg \mathsf{V}_i$ and $\neg \mathsf{C}_i \sqcap \mathsf{C}'_i \sqsubseteq \forall \mathsf{R}.\mathsf{V}_i$, for $1 \leq i \leq n$, hold under the interpretation $\mathcal{I}$. We know that $\forall 0 < l \leq j, (s_{l-1}, s_l) \in R$. According to the definition of model description, it means that $\forall 0 < i \leq n$, $s_{l-1} \models c_i$ implies $s_l \models \neg v_i$ and $s_{l-1} \models \neg c_i \wedge c'_i$ implies $s_l \models v_i$. By Lemma 5 we get that $\sigma_{l-1} \in \mathsf{C}_i^{\mathcal{I}}$ implies $\sigma_l \notin \mathsf{V}_i^{\mathcal{I}}$ and $\sigma_{l-1} \in (\Delta^{\mathcal{I}} \setminus \mathsf{C}_i^{\mathcal{I}}) \cap \mathsf{C}_i'^{\mathcal{I}}$ implies $\sigma_l \in \mathsf{V}_i^{\mathcal{I}}$. Since no pairs other than (σ_{l-1}, σ_l) belong to $\mathsf{R}^{\mathcal{I}}$ in the interpretation $\mathcal{I}$, the inclusions hold.
- $\mathsf{C}_\varphi^{\mathcal{I}}$ is not empty: We shall show that $\sigma_j \in \mathsf{C}_\varphi^{\mathcal{I}}$. Recall that

  $$\mathsf{C}_\varphi \equiv \neg \mathsf{B} \sqcap (\mathsf{S}_0 \sqcup \mathsf{S}_1 \sqcup ... \sqcup \mathsf{S}_k)$$

  and therefore under the interpretation $\mathcal{I}$,

  $$\mathsf{C}_\varphi^{\mathcal{I}} = (\Delta^{\mathcal{I}} \setminus \mathsf{B}^{\mathcal{I}}) \cap (\mathsf{S}_0^{\mathcal{I}} \cup \mathsf{S}_1^{\mathcal{I}} \cup ... \cup \mathsf{S}_k^{\mathcal{I}})$$

  Since $\mathsf{S}_j = \{\sigma_j\}$, we get that $\sigma_j \in (\mathsf{S}_0^{\mathcal{I}} \cup \mathsf{S}_1^{\mathcal{I}} \cup ... \cup \mathsf{S}_k^{\mathcal{I}})$. Since $s_j \models \neg b$, we get by Lemma 5 that $\sigma_j \notin \mathsf{B}^{\mathcal{I}}$. Thus $\sigma_j \in \Delta^{\mathcal{I}} \setminus \mathsf{B}^{\mathcal{I}}$, and therefore $\sigma_j \in \mathsf{C}_\varphi^{\mathcal{I}}$.

($\Longleftarrow$) Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation showing that $\mathcal{T}_{MD}^k \models_{dl} \mathsf{C}_\varphi$ is consistent. We have to show that $M_{MD}^k \not\models \varphi$. Since $\mathsf{C}_\varphi^{\mathcal{I}} = (\Delta^{\mathcal{I}} \setminus \mathsf{B}^{\mathcal{I}}) \cap (\mathsf{S}_0^{\mathcal{I}} \cup \mathsf{S}_1^{\mathcal{I}} \cup ... \cup \mathsf{S}_k^{\mathcal{I}})$ is not empty in $\mathcal{I}$, it must be the case that for some j, $0 \leq j \leq k$, $(\Delta^{\mathcal{I}} \setminus \mathsf{B}^{\mathcal{I}}) \cap \mathsf{S}_j^{\mathcal{I}}$ is not empty. Let σ_j be an element in $(\Delta^{\mathcal{I}} \setminus \mathsf{B}^{\mathcal{I}}) \cap \mathsf{S}_j^{\mathcal{I}}$. Then $\sigma_j \in (\Delta^{\mathcal{I}} \setminus \mathsf{B}^{\mathcal{I}})$ and also $\sigma_j \in \mathsf{S}_j^{\mathcal{I}}$.

Since $\mathcal{T}_{MD}^k$ includes the concept inclusion $\mathsf{S}_j \sqsubseteq \exists \mathsf{R}^{-}.\mathsf{S}_{j-1}$, and $\mathsf{S}_j^{\mathcal{I}}$ is not empty, we deduce that $\mathsf{S}_{j-1}^{\mathcal{I}}$ is not empty, and that $\exists \sigma_{j-1} \in \mathsf{S}_{j-1}^{\mathcal{I}}$, such that $(\sigma_{j-1}, \sigma_j) \in \mathsf{R}^{\mathcal{I}}$. By similar considerations, there must exist a sequence of elements $\sigma_0, ..., \sigma_j \in \Delta^{\mathcal{I}}$, such that for $0 \leq l < j$, $(\sigma_l, \sigma_{l+1}) \in \mathsf{R}^{\mathcal{I}}$, and $\sigma_0 \in \mathsf{S}_0^{\mathcal{I}}$. We define a sequence of states $s_0, ..., s_j$ from M_{MD} according to the function F from Definition 4: $F(\sigma_l) = s_l$.

We need to prove that for $0 \leq l < j$, $(s_l, s_{l+1}) \in R$, $s_0 \models I$, $s_j \models \neg b$. These follow easily from Lemma 5 as shown below.

- $s_0 \models I$. Recall that the concept S_0 corresponds to the condition I of the model M_{MD}, which is a Boolean combination of the variables v_i. Thus since $\sigma_0 \in \mathsf{S}_0^{\mathcal{I}}$ we get by Lemma 5 that $s_0 \models I$.
- $s_j \models \neg b$. As shown above, $\sigma_j \in (\Delta^{\mathcal{I}} \setminus \mathsf{B}^{\mathcal{I}})$ and therefore $\sigma_j \notin \mathsf{B}^{\mathcal{I}}$. By Lemma 5, $s_j \not\models b$, that is $s_j \models \neg b$.
- $(s_l, s_{l+1}) \in R$. Since all concept inclusions of $\mathcal{T}_{MD}^k$ hold under the interpretation $\mathcal{I}$, we know that $\forall 1 \leq i \leq n$,
  $\mathsf{C}_i^{\mathcal{I}} \subseteq \{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}}, (x,y) \in \mathsf{R}^{\mathcal{I}} \rightarrow y \in \Delta^{\mathcal{I}} \setminus \mathsf{V}_i\}$ and also
  $(\Delta^{\mathcal{I}} \setminus \mathsf{C}_i^{\mathcal{I}}) \cap \mathsf{C}_i'^{\mathcal{I}} \subseteq \{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}}, (x,y) \in \mathsf{R}^{\mathcal{I}} \rightarrow y \in \mathsf{V}_i\}$.

Thus if $\sigma_l \in \mathsf{C}_i^{\mathcal{I}}$ it must be the case that $\sigma_{l+1} \notin \mathsf{V}_i^{\mathcal{I}}$, and similarly if $\sigma_l \in (\neg\mathsf{C}_i^{\mathcal{I}} \cap \mathsf{C}_i'^{\mathcal{I}})$ it must be the case that $\sigma_{l+1} \in \mathsf{V}_i^{\mathcal{I}}$. Because of the correspondence between σ_l and s_l, we have that $s_l \models c_i$ implies $s_{l+1} \models \neg v_i$ and $s_l \models c_i' \wedge \neg c_i$ implies $s_{l+1} \models v_i$. Thus by definition, $(s_l, s_{l+1}) \in R$.

This concludes the proof. □

3.4 Experiments

We implemented our method and experimented with it using the Description Logic reasoner *FACT*$^{++}$ [13]. We present two sets of results. In Table 1 we compare our method with the one reported in [3], where a different encoding of model checking in DL is described. The method in [3] applies a *forward* search, as opposed to the backward search proposed in this paper. We compare the two methods on a very simple model, parameterized so we can run it with increasing numbers of state variables. For the backward method we chose the bound to be 20, although the diameter of the model is only 3. The safety formula we verified is satisfied in all three models. Table 1 shows the number of concepts and concept inclusions needed to describe the model for each method, and the time in seconds it takes to execute. We set a timeout of 1200 seconds. The results demonstrate that the backward search, described in this paper, significantly outperforms the forward search approach.

Table 1. Forward vs. backward model checking using DL

	Forward Search			Backward Search		
Model Size	concepts	inclusions	time	concepts	inclusions	time
10	32	104	0.07	22	25	0
20	62	204	> 1200	32	40	0.01
50	152	514	> 1200	62	110	0.02

In Table 2 we present results comparing our method (backward search) with two other symbolic model checking methods. We used *NuSMV* [6], version 2.4.1, as a BDD-based model checker. We used the BMC mode of NuSMV, that invoked *zChaff* [17] as a SAT solver for bounded model checking. The model we use for comparison is

Table 2. DL Model Checking Vs. SMV and SAT

Model Size	Bound	DL-Backward	NuSMV	zChaff
85	5	4	> 1200	1
85	10	> 1200	> 1200	0.9
272	5	202	> 1200	1.2
272	10	> 1200	> 1200	3.9
425	5	335	> 1200	2.4
425	10	> 1200	> 1200	6.7

derived from the NuSMV example “dme1-16”, taken from [18] and parameterized to have different numbers of cells. The formula verified is a safety property that holds in the model. We did not attempt to optimize the run of any tool (many options are available), but rather, ran them in their default mode. Although our method performs better than NuSMV on the given examples, it still falls far behind the performance of zChaff. In particular, we note that the DL method seems to be very sensitive to the bound k on the depth of the search.

4 Related Work

A connection between knowledge-based reasoning and model checking has been explored before. Gottlob et al in [11,12] analyzed the expressive power of *Datalog* statements, and compared them to known temporal logics. Sahasrabudhe in [21] has performed model checking of telephony feature interactions by using SQL on an explicit state representation of the model, and has compared the results with model checking a similar explicit state representation using the model checker SMV [16]. Both Sahasrabudhe and Gottlob et al however, used an explicit representation of the model, as opposed to the symbolic representation that we propose. This difference is crucial, since in many cases the Kripke structure for the model is too big to be built, and symbolic methods must be used. Dovier and Quintarelli in [10] were interested in the opposite direction: they translated a knowledge-base into a Kripke structure, and a query into a temporal logic formula. They then used a model checker to make inferences about the knowledge-base.

In a previous paper [3] we gave a first formulation of a model checking problem as a terminology in Description logic. The formulation there has some advantages over the current: it performs unbounded model checking rather than bounded model checking that we propose here; it supports safety as well as liveness formulas, and it uses the simpler dialect $\mathcal{ALC}$, rather than $\mathcal{ALCI}$ that we use here. However, the terminology built for a given model description employs three times as many concepts and five times as many concept inclusions. In addition, the reasoning involved synchronizing the progress of the different state variables. Thus the performance of that method, as shown in Section 3.4, was much worse than the one presented in this paper.

Finally, a compact representation of a BMC problem, with size similar to the one described in this paper, is also achieved when presenting the model description as a Quantified Boolean Formula (QBF). Recent activity in this area [9,15] suggest though, that this too does not perform as well as SAT solvers.

5 Conclusions and Future Work

We have shown how Description Logic can serve as a natural setting for representing a BMC problem, avoiding the need to unfold the model. Thus for a given model description MD and a bound k, the size of the representation is $|MD| + k$, as opposed to $|MD| \times k$ when translating MD to a propositional formula. Experimental results show a significant improvement over a different method of model checking using DL, and comparable performance with BDD-based model checking.

While performance is still not competitive with the SAT-based approach, we believe that model checking using DL reasoning merits further research. One future direction is to better exploit *absorption* [14]. Absorption is a pre-processing technique that allows the elimination of some forms of concept inclusions by converting them into augmented concept definitions. Our current translation into DL does not allow absorption for most of the concept inclusions.

Acknowledgements

We thank Dmitry Tsarkov for his support in the installation of the $FaCT^{++}$ Description Logic reasoner. We thank Vlad Ciubotariu for his help in running NuSMV. Finally, we would like to acknowledge the financial support provided by NSERC of Canada.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
2. Beer, I., Ben-David, S., Landver, A.: On-the-fly model checking of RCTL formulas. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, Springer, Heidelberg (1998)
3. Ben-David, S., Trefler, R., Weddell, G.: Model checking the basic modalities of CTL with description logic. In: Proc. International Workshop on Description Logics, pp. 223–230 (2006)
4. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) ETAPS 1999 and TACAS 1999. LNCS, vol. 1579, Springer, Heidelberg (1999)
5. Bryant, R.: Graph-based algorithms for boolean function manipulation. In IEEE Transactions on Computers, vol. c-35(8) (1986)
6. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new symbolic model verifier. In: Computer Aided Verification, pp. 495–499, (July 1999)
7. Clarke, E., Emerson, E.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logics of Programs. LNCS, vol. 131, Springer, Heidelberg (1982)
8. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. The MIT Press, Cambridge (2000)
9. Dershowitz, N., Hanna, Z., Katz, J.: Bounded model checking with QBF. In: Eighth International Conference on Theory and Applications of Satisfiability Testing, pp. 408–414, (June 2005)
10. Dovier, A., Quintarelli, E.: Model checking based data retrieval. In: Ghelli, G., Grahne, G. (eds.) DBPL 2001. LNCS, vol. 2397, Springer, Heidelberg (2002)
11. Gottlob, G., Grädel, E., Veith, H.: Linear Time Datalog for Branching Time Logic (chapter 19). In: Minker, J. (ed.) Logic-Based Artificial Intelligence, Kluwer, Boston (2000)
12. Gottlob, G., Grädel, E., Veith, H.: Datalog LITE: a deductive query language with linear time model checking. Computational Logic 3(1), 42–79 (2002)
13. Horrocks, I.: The FaCT system. Lecture Notes in Computer Science 1397, 307–312 (1998)
14. Horrocks, I., Tobies, S.: Reasoning with axioms: Theory and practice. In: Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000), pp. 285–296 (2000)
15. Jussila, T., Biere, A.: Compressing bmc encodings with QBF. In: Fourth International Workshop on Bounded Model Checking, pp. 27–39, (August 2006)

16. McMillan, K.: Symbolic Model Checking. Kluwer Academic Publishers, Norwell (1993)
17. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: 38th Design Automation Conference, pp. 530–535, (June 2001)
18. NuSMV examples collection, `http://nusmv.irst.itc.it/examples/examples.html`
19. Pnueli, A.: The temporal logic of programs. In: 18th IEEE Symposium on Foundation of Computer Science, pp. 46–57 (1977)
20. Quielle, J., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: *5th International Symposium on Programming* (1982)
21. Sahasrabudhe, M.: SQL-based CTL model checking for telephony feature interactions. In: A Master Thesis, Univesity of Waterloo, Ontario, Canada (2004)

Tableau Systems for Logics of Subinterval Structures over Dense Orderings

Davide Bresolin[1], Valentin Goranko[2], Angelo Montanari[3], and Pietro Sala[3]

[1] Department of Computer Science, University of Verona,
Verona, Italy
bresolin@sci.univr.it

[2] School of Mathematics, University of the Witwatersrand,
Johannesburg, South Africa
goranko@maths.wits.ac.za

[3] Department of Mathematics and Computer Science,
University of Udine, Udine, Italy
{montana,sala}@dimi.uniud.it

Abstract. We construct a sound, complete, and terminating tableau system for the interval temporal logic $\mathrm{D}_{\sqsubset\!\!\cdot}$ interpreted in interval structures over dense linear orderings endowed with *strict* subinterval relation (where both endpoints of the sub-interval are strictly inside the interval). In order to prove the soundness and completeness of our tableau construction, we introduce a kind of finite pseudo-models for our logic, called $\mathrm{D}_{\sqsubset\!\!\cdot}$-structures, and show that every formula satisfiable in $\mathrm{D}_{\sqsubset\!\!\cdot}$ is satisfiable in such pseudo-models, thereby proving small-model property and decidability in PSPACE of $\mathrm{D}_{\sqsubset\!\!\cdot}$, a result established earlier by Shapirovsky and Shehtman by means of filtration. We also show how to extend our results to the interval logic $\mathrm{D}_{\sqsubset}$ interpreted over dense interval structures with *proper* (irreflexive) subinterval relation, which differs substantially from $\mathrm{D}_{\sqsubset\!\!\cdot}$ and is generally more difficult to analyze. Up to our knowledge, no complete deductive systems and decidability results for $\mathrm{D}_{\sqsubset}$ have been proposed in the literature so far.

1 Introduction

Interval-based temporal logics provide a natural framework for temporal representation and reasoning. However, while many tableau systems have been developed for point-based temporal logics, few tableau systems have been constructed for interval temporal logics [3,6,9], as these are generally more complex. Even fewer tableau systems for interval logics provide decision procedures – a reflection of the general phenomenon of undecidability of interval-based temporal logics. Notable recent exceptions are [4,5,6,7,8].

In this paper we consider *interval temporal logics interpreted in interval structures over dense linear orderings endowed with subinterval relations.* These structures arise quite naturally and appear deceivingly simple, while actually they are not. Perhaps for that reason they have been studied very little yet, and we are aware of very few publications containing any representation results, complete

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 73–89, 2007.

deductive systems, or decidability results for subinterval structures and logics. The only known simple case is the logic $\mathrm{D}_{\sqsubseteq}$, where the reflexive subinterval relation is considered, which has been proved to be equivalent to the modal logic S4 of reflexive and transitive frames when interpreted over dense orderings in [1]. Neither of the two (irreflexive) cases we take into consideration in this work reduces to K4. Besides the purely mathematical attraction arising from the combination of conceptual simplicity with technical challenge, the study of subinterval structures and logics turns out to be important because they provide, together with the neighborhood interval logics, the currently most intriguing and under-explored fragments of Halpern-Shoham's interval logic HS [10]. They occupy a region on the very borderline between decidability and undecidability of propositional interval logics, and since decidability results in that area are preciously scarce, complete and terminating tableau systems like those constructed in the paper are of particular interest. (It should be noted that the decidability results obtained here do not follow from the decidability of the MSO over the rational order, because the semantics of the considered interval logics is essentially dyadic second-order).

Here we focus our attention on the logic $\mathrm{D}_{\sqsubset\!\!\!\cdot\,}$, corresponding to the case of strict subinterval relation (where both endpoints of the subinterval are strictly inside the interval) over the class of dense linear orderings. These subinterval structures turn out to be intimately related (essentially, interdefinable) with Minkowski space-time structures. The relations between the logic $\mathrm{D}_{\sqsubset\!\!\!\cdot\,}$ and the logic of Minkowski space-time were studied by Shapirovsky and Shehtman in [12]. They established a sound and complete axiomatic system for $\mathrm{D}_{\sqsubset\!\!\!\cdot\,}$ and proved its decidability and PSPACE-completeness by means of a non-trivial filtration technique [11,12]. In this paper, we construct a sound, complete, and terminating tableau system for $\mathrm{D}_{\sqsubset\!\!\!\cdot\,}$. In order to prove the soundness and completeness of our tableau construction, we introduce a kind of finite pseudo-models for $\mathrm{D}_{\sqsubset\!\!\!\cdot\,}$, called $\mathrm{D}_{\sqsubset\!\!\!\cdot\,}$-structures, and show that every formula satisfiable in $\mathrm{D}_{\sqsubset\!\!\!\cdot\,}$ is satisfiable in such pseudo-models, thereby proving small-model property and decidability in PSPACE of $\mathrm{D}_{\sqsubset\!\!\!\cdot\,}$. Moreover, we extend our results to the case of the interval logic $\mathrm{D}_{\sqsubset}$ interpreted in interval structures over dense linear orderings with *proper* (irreflexive) subinterval relation, which differs substantially from $\mathrm{D}_{\sqsubset\!\!\!\cdot\,}$ and is generally more difficult to analyze. Up to our knowledge, no decidability or completeness results for deductive systems for that logic have been proposed yet.

2 Syntax and Semantics of $\mathrm{D}_{\sqsubset\!\!\!\cdot\,}$

Let $\mathbb{D} = \langle D, < \rangle$ be a dense linear order. An *interval* over $\mathbb{D}$ is an ordered pair $[b, e]$, where $b < e$. We denote the set of all intervals over $\mathbb{D}$ by $\mathbb{I}(\mathbb{D})^-$ (we use the superscript $^-$ to indicate that point-intervals $[b, b]$ are excluded).

We consider three *subinterval relations*: the *reflexive subinterval relation* (denoted by $\sqsubseteq$), defined by $[d_k, d_l] \sqsubseteq [d_i, d_j]$ iff $d_i \leq d_k$ and $d_l \leq d_j$, the *proper (or irreflexive) subinterval relation* (denoted by $\sqsubset$), defined by $[d_k, d_l] \sqsubset [d_i, d_j]$ iff $[d_k, d_l] \sqsubseteq [d_i, d_j]$ and $[d_k, d_l] \neq [d_i, d_j]$, and the *strict subinterval relation* (denoted by $\sqsubset\!\!\!\cdot\,$), defined by $[d_k, d_l]\sqsubset\!\!\!\cdot\,[d_i, d_j]$ iff $d_i < d_k$ and $d_l < d_j$. In this paper we will only deal with the latter two cases, beginning with $\sqsubset\!\!\!\cdot\,$.

The language of the modal logic $\mathrm{D}_{\sqsubset\!\!\!\cdot}$ of interval structures with strict subinterval relation consists of a set $\mathcal{AP}$ of propositional letters, the propositional connectives $\neg$ and $\vee$, and the modal operator $\langle D\rangle$. The other propositional connectives, as well as the logical constants $\top$ (*true*) and $\bot$ ($false$) and the dual modal operator $[D]$, are defined as usual. The formulae of $\mathrm{D}_{\sqsubset\!\!\!\cdot}$ are defined as usual: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle D\rangle\varphi$.

The semantics of $\mathrm{D}_{\sqsubset\!\!\!\cdot}$ is given in *interval models* $\mathbf{M}^- = \langle \mathbb{I}(\mathbb{D})^-, \sqsubset\!\!\!\cdot\,, \mathcal{V}\rangle$. The *valuation function* $\mathcal{V} : \mathcal{AP} \mapsto 2^{\mathbb{I}(\mathbb{D})^-}$ assigns to every propositional variable p the set of intervals $\mathcal{V}(p)$ over which p holds[1]. The semantics of $\mathrm{D}_{\sqsubset\!\!\!\cdot}$ is recursively defined by the satisfiability relation $\Vdash$ as follows:

- for every propositional variable $p \in \mathcal{AP}$, $\mathbf{M}^-, [d_i, d_j] \Vdash p$ iff $[d_i, d_j] \in \mathcal{V}(p)$;
- $\mathbf{M}^-, [d_i, d_j] \Vdash \neg\psi$ iff $\mathbf{M}^-, [d_i, d_j] \not\Vdash \psi$;
- $\mathbf{M}^-, [d_i, d_j] \Vdash \psi_1 \vee \psi_2$ iff $\mathbf{M}^-, [d_i, d_j] \Vdash \psi_1$ or $\mathbf{M}^-, [d_i, d_j] \Vdash \psi_2$;
- $\mathbf{M}^-, [d_i, d_j] \Vdash \langle D\rangle\psi$ iff $\exists [d_k, d_l] \in \mathbb{I}(D)^-$ such that $[d_k, d_l]\sqsubset\!\!\!\cdot\,[d_i, d_j]$ and $\mathbf{M}^-, [d_k, d_l] \Vdash \psi$.

A $\mathrm{D}_{\sqsubset\!\!\!\cdot}$-formula is $\mathrm{D}_{\sqsubset\!\!\!\cdot}$-*satisfiable* if it is true in some interval in some interval model and it is $\mathrm{D}_{\sqsubset\!\!\!\cdot}$-*valid* if it is true in every interval in every interval model. The logic $\mathrm{D}_{\sqsubset}$ has the same language as $\mathrm{D}_{\sqsubset\!\!\!\cdot}$, but it is interpreted in *irreflexive interval models* $\mathbf{M}^- = \langle \mathbb{I}(\mathbb{D})^-, \sqsubset, \mathcal{V}\rangle$.

In [12] a logic $\mathbf{L}_1$ was considered and completely axiomatized as follows:

$$\mathbf{L}_1 = \mathbf{K}_4 + \Diamond\top + \Diamond p_1 \wedge \Diamond p_2 \rightarrow \Diamond(\Diamond p_1 \wedge \Diamond p_2).$$

That logic was shown in [12] to be essentially the logic of the strict subinterval structure over the rational ordering $(\mathbb{Q}, <)$. In the next section we show that the latter coincides with the logic $\mathrm{D}_{\sqsubset\!\!\!\cdot}$, and therefore $\mathrm{D}_{\sqsubset\!\!\!\cdot}$ and $\mathbf{L}_1$ turn out to be the same.

3 Structures for $\mathrm{D}_{\sqsubset\!\!\!\cdot}$

To devise a decision procedure for $\mathrm{D}_{\sqsubset\!\!\!\cdot}$, we first interpret it over a special class of graphs, that we call $\mathrm{D}_{\sqsubset\!\!\!\cdot}$-graphs. We will prove that a $\mathrm{D}_{\sqsubset\!\!\!\cdot}$-formula is satisfiable in a dense interval structure if and only if it is satisfiable in a model over a $\mathrm{D}_{\sqsubset\!\!\!\cdot}$-graph. Furthermore, it will turn out that this is equivalent to satisfiability in an interval structure over the interval $[0, 1]$ of the rational line.

We begin by introducing the key notion of φ-*atom* and the relation D_φ connecting φ-atoms. Given a $\mathrm{D}_{\sqsubset\!\!\!\cdot}$-formula φ, let $\mathrm{CL}(\varphi)$ be the *closure of* φ, defined as the set of all sub-formulae of φ and their negations. A φ-atom is defined as follows.

[1] We emphasize that formulae are evaluated only *relative to intervals* and not to points. Thus, intervals are regarded as primitive entities and the formulae only express properties of intervals and their subintervals. In particular, no assumptions are made relating truth of an atomic formula at an interval to its truth at subintervals of that interval.

Definition 1. *Given a* $\mathrm{D}_{\sqsubset}$*-formula* φ*, a* φ-atom A *is a subset of* $\mathrm{CL}(\varphi)$ *such that (i) for every* $\psi \in \mathrm{CL}(\varphi)$*,* $\psi \in A$ *if and only if* $\neg\psi \notin A$ *and (ii) for every* $\psi_1 \vee \psi_2 \in \mathrm{CL}(\varphi)$*,* $\psi_1 \vee \psi_2 \in A$ *if and only if* $\psi_1 \in A$ *or* $\psi_2 \in A$*.*

We denote the set of all φ-atoms by $\mathcal{A}_\varphi$.

Definition 2. *Let* D_φ *be a binary relation over* $\mathcal{A}_\varphi$ *such that, for every pair of* φ*-atoms* $A, A' \in \mathcal{A}_\varphi$*,* $A \, D_\varphi \, A'$ *holds if and only if* $\psi \in A'$ *and* $[D]\psi \in A'$ *for every formula* $[D]\psi \in A$*.*

A φ*-atom* A *is* reflexive *if* $A D_\varphi A$ *holds, otherwise it is* irreflexive*.*

When φ is clear from the context, φ-atoms will be called simply ‘atoms’.

Given a directed graph $\mathbb{G} = \langle V, E\rangle$, a vertex $v \in V$ is *reflexive* if the edge (v, v) belongs to E, otherwise v is an *irreflexive vertex*.

Definition 3. *A finite directed graph* $\mathbb{G} = \langle V, E\rangle$ *is a* $\mathrm{D}_{\sqsubset}$-graph *if (and only if) the following conditions hold:*

1. *there exists an irreflexive vertex* $v_0 \in V$*, called the* root *of* $\mathbb{G}$*, such that any other vertex* $v \in V$ *is reachable from it;*
2. *every irreflexive vertex* $v \in V$ *has a unique successor* v_D*, which is reflexive;*
3. *every successor of a reflexive vertex* v*, different from* v*, is irreflexive.*

A $\mathrm{D}_{\sqsubset}$-graph is depicted in Figure 1. $\mathrm{D}_{\sqsubset}$-graphs are finite by definition, but they may include loops involving irreflexive vertices.

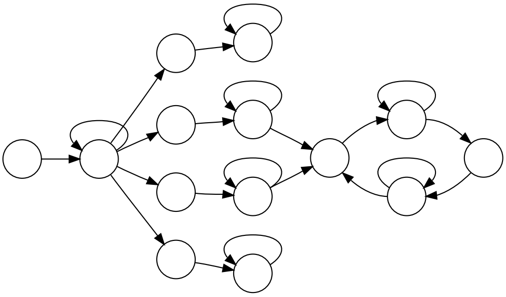


Fig. 1. An example of $\mathrm{D}_{\sqsubset}$-graph

A $\mathrm{D}_{\sqsubset}$-structure is a $\mathrm{D}_{\sqsubset}$-graph paired with a labeling function that assigns an atom to every vertex in the graph. It is formally defined as follows.

Definition 4. *A* $\mathrm{D}_{\sqsubset}$-structure *is a pair* $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ *where* $\langle V, E\rangle$ *is a* $\mathrm{D}_{\sqsubset}$*-graph and* $\mathcal{L} : V \to \mathcal{A}_\varphi$ *is a* labeling function *that assigns to every vertex* $v \in V$ *an atom* $\mathcal{L}(v)$ *such that* $\mathcal{L}(v) \, D_\varphi \, \mathcal{L}(v')$ *for every edge* $(v, v') \in E$*. Let* v_0 *be the root of* $\langle V, E\rangle$*. If* $\varphi \in \mathcal{L}(v_0)$*, we say that* $\mathbf{S}$ *is a* $\mathrm{D}_{\sqsubset}$-structure for φ*.*

$\mathrm{D}_{\sqsubset}$-structures can be viewed as ‘pseudo-models’ for $\mathrm{D}_{\sqsubset}$. Formulae devoid of temporal operators are satisfied by the definition of φ-atom; moreover, $[D]$-formulae are satisfied by the definition of D_φ. To guarantee the satisfiability of $\langle D\rangle$-formulae, we introduce the notion of *fulfilling* $\mathrm{D}_{\sqsubset}$*-structures.*

Definition 5. *A* $\mathrm{D}_{\sqsubset}$*-structure* $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ *is* fulfilling *if and only if for every vertex* $v \in V$ *and every formula* $\langle D\rangle\psi \in \mathcal{L}(v)$*, there exists a descendant (i.e., vertex reachable by a path of successors)* v' *of* v *such that* $\psi \in \mathcal{L}(v')$*.*

Theorem 1. *Let* φ *be a* $\mathrm{D}_{\sqsubset}$*-formula which is satisfied in a dense interval model. Then, there exists a fulfilling* $\mathrm{D}_{\sqsubset}$*-structure* $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ *such that* $\varphi \in \mathcal{L}(v_0)$*, where* v_0 *is the root of* $\langle V, E\rangle$*.*

Proof. Let $\mathbf{M} = \langle \mathbb{I}(\mathbb{D})^-, \sqsubset, \mathcal{V}\rangle$ be a dense interval model and let $[b_0, e_0] \in \mathbb{I}(\mathbb{D})^-$ be an interval such that $\mathbf{M}, [b_0, e_0] \Vdash \varphi$. We recursively build a fulfilling $\mathrm{D}_{\sqsubset}$-structure $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ for φ as follows.

We start with the one-node graph $\langle\{v_0\}, \emptyset\rangle$ and the labeling function $\mathcal{L}$ such that $\mathcal{L}(v_0) = \{\psi \in \mathrm{CL}(\varphi) : \mathbf{M}, [b_0, e_0] \Vdash \psi\}$.

Next, for every formula $\langle D\rangle\psi \in \mathcal{L}(v_0)$ we pick up an interval $[b_\psi, e_\psi]$ such that $[b_\psi, e_\psi] \sqsubset [b_0, e_0]$ and $\mathbf{M}, [b_\psi, e_\psi] \Vdash \psi$; it exists by definition of $\mathcal{L}(v_0)$.

Then, since $\mathbb{D}$ is a dense ordering and $\mathrm{CL}(\varphi)$ is a finite set of formulae, we can find two intervals $[b_1, e_1]$ and $[b_2, e_2]$ such that:

- $[b_2, e_2] \sqsubset [b_1, e_1] \sqsubset [b_0, e_0]$;
- for every $\langle D\rangle\psi \in \mathcal{L}(v_0)$, $[b_\psi, e_\psi] \sqsubset [b_2, e_2]$;
- $[b_1, e_1]$ and $[b_2, e_2]$ satisfy the same formulae of $\mathrm{CL}(\varphi)$.

Since $\mathbf{M}$ is a model and $[b_\psi, e_\psi] \sqsubset [b_2, e_2]$ for every interval $[b_\psi, e_\psi]$, $[b_2, e_2]$ satisfies $\langle D\rangle\psi$ for every $\langle D\rangle\psi \in \mathcal{L}(v_0)$. Moreover, since $[b_1, e_1]$ and $[b_2, e_2]$ satisfy the same formulae of $\mathrm{CL}(\varphi)$ and $[b_2, e_2] \sqsubset [b_1, e_1]$, for every $[D]\psi \in \mathrm{CL}(\varphi)$, if $[b_2, e_2]$ satisfies $[D]\psi$, then it satisfies ψ as well.

Accordingly, we add a new (reflexive) vertex v_D and the edges (v_0, v_D) and (v_D, v_D) to the graph and we label v_D by $\mathcal{L}(v_D) = \{\xi \in \mathrm{CL}(\varphi) : \mathbf{M}, [b_2, e_2] \Vdash \xi\}$. Furthermore, for every interval $[b_\psi, e_\psi]$, we add a new (irreflexive) vertex v_ψ, together with the edge $(v_D, v_\psi) \in E$, and we label it by $\mathcal{L}(v_\psi) = \{\xi \in \mathrm{CL}(\varphi) : \mathbf{M}, [b_\psi, e_\psi] \Vdash \xi\}$. Finally, to obtain a $\mathrm{D}_{\sqsubset}$-structure for φ, we recursively apply the above construction to the vertices $v_{\psi_1}, \ldots, v_{\psi_k}$.

To keep the construction finite, whenever the above procedure requests us to introduce a successor v' of a reflexive (resp., irreflexive) node $v \in V$, but there exists an irreflexive (resp., reflexive) node $w \in V$ such that $\mathcal{L}(w) = \mathcal{L}(v')$, we replace the addition of the node v' with the addition of an edge from v to w. Since the set of atoms is finite, this guarantees the termination of the construction process. □

Now, let $\mathbf{S}$ be a fulfilling $\mathrm{D}_{\sqsubset}$-structure for a formula φ. We will prove that φ is satisfiable in a dense interval structure. Moreover, we will show that such a structure can be constructed on the interval $[0, 1]$ of the rational line. To begin with, we define a function $f_{\mathbf{S}}$ connecting intervals in $\mathbb{I}([0, 1])^-$ with vertices in $\mathbf{S}$. Such a function will allow us to define a model for φ.

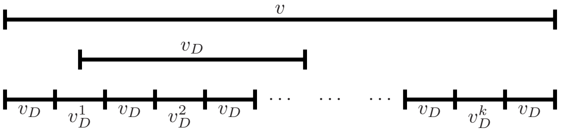
Definition 6. *Let* $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ *be a* $\mathrm{D}_{\sqsubset}$*-structure. The function* $f_{\mathbf{S}} : \mathbb{I}([0, 1])^- \mapsto V$ *is recursively defined as follows:*

– *$f_{\mathbf{S}}([0,1]) = v_0$;*

– *let $[b,e]$ be an interval such that $f_{\mathbf{S}}([b,e]) = v$ and $f_{\mathbf{S}}$ has not yet been defined for any subinterval of $[b,e]$. Let v_D be the unique reflexive successor of v if v is irreflexive, and let $v_D = v$ otherwise. There are two alternatives:*

1. v_D has no successors other than itself. *In such a case, we put $f_{\mathbf{S}}([b',e']) = v_D$ for every proper subinterval $[b',e']$ of $[b,e]$.*
2. v_D has at least one successor different from itself. *Let $v_D^1, \ldots, v_D^k$ be the successors of v_D different from v_D. We consider the intervals defined by the points $b, b+p, b+2p \ldots, b+2kp, b+(2k+1)p = e$, with $p = \frac{e-b}{2k+1}$. The function $f_{\mathbf{S}}$ over such intervals is defined as follows:*
   - *for every $i = 1, \ldots, k$, we put $f_{\mathbf{S}}([b+(2i-1)p, b+2ip]) = v_D^i$.*
   - *for every $i = 0, \ldots, k$, we put $f_{\mathbf{S}}([b+2ip, b+(2i+1)p]) = v_D$.*

   We complete the construction by putting $f_{\mathbf{S}}([b',e']) = v_D$ for every subinterval $[b',e']$ of $[b,e]$ which is not a subinterval of any of the intervals $[b+ip, b+(i+1)p]$. The resulting structure is depicted below.



It is easy to show that $f_{\mathbf{S}}$ satisfies the following properties.

Lemma 1

1. *For every pair of intervals $[b,e], [b',e'] \in \mathbb{I}([0,1])^-$ such that $[b',e'] \sqsubset [b,e]$, $f_{\mathbf{S}}([b',e'])$ is reachable from $f_{\mathbf{S}}([b,e])$.*
2. *For every interval $[b,e] \in \mathbb{I}([0,1])^-$, if $f_{\mathbf{S}}([b,e]) = v$ and v' is reachable from v, then there exists $[b',e'] \sqsubset [b,e]$ such that $f_{\mathbf{S}}([b',e']) = v'$.*

Remark 1. The lemma above means that the mapping $f_{\mathbf{S}}$ defined above is a bounded morphism from $\mathbb{I}([0,1])^-$ to (in fact, onto) V, and the theorem below follows from a general result in modal logic stating truth-preservation under bounded morphisms, see e.g. [2]. However, since we do not regard $\mathrm{D}_{\sqsubset}$-structures a standard models for our logic, we will not make use of this reference, but will give a direct proof instead.

For any fulfilling $\mathrm{D}_{\sqsubset}$-structure $\mathbf{S}$ for φ, let $\mathbf{M_S}$ be the triplet $\langle \mathbb{I}([0,1])^-, \sqsubset, \mathcal{V} \rangle$, where $\mathcal{V}(p) = \{[b,e] : p \in \mathcal{L}(f_{\mathbf{S}}([b,e]))\}$ for every $p \in \mathcal{AP}$. It turns out that $\mathbf{M_S}$ is a model for φ.

Theorem 2. *Let $\mathbf{S}$ be a fulfilling $\mathrm{D}_{\sqsubset}$-structure for φ. Then $\mathbf{M_S}, [0,1] \Vdash \varphi$.*

Proof. We prove that for every interval $[b,e] \in \mathbb{I}([0,1])^-$ and every formula $\psi \in \mathrm{CL}(\varphi)$, $\mathbf{M_S}, [b,e] \Vdash \psi$ if and only if $\psi \in \mathcal{L}(f_{\mathbf{S}}([b,e])$. The proof is by induction on the structure of the formula:

- the case of propositional letters as well as those of Boolean connectives are straightforward and thus omitted;
- let $\psi = \langle D \rangle \xi$, and suppose that $\psi \in \mathcal{L}(f_{\mathbf{S}}([b,e]))$. Since $\mathbf{S}$ is fulfilling, there exists a vertex v', which is reachable from $f_{\mathbf{S}}([b,e])$, such that $\xi \in \mathcal{L}(v')$.

By Lemma 1, there exists $[b', e'] \sqsubset [b, e]$ such that $f_{\mathbf{S}}([b', e']) = v'$. By the inductive hypothesis, $\mathbf{M}_{\mathbf{S}}, [b', e'] \Vdash \xi$ and thus $\mathbf{M}_{\mathbf{S}}, [b, e] \Vdash \langle D \rangle \xi$.

To prove the converse implication, suppose for reductio ad absurdum that $\mathbf{M}_{\mathbf{S}}, [b, e] \Vdash \langle D \rangle \xi$ but $\langle D \rangle \xi \notin \mathcal{L}(f_{\mathbf{S}}([b, e]))$. By the definition of φ-atom, this implies that $[D]\neg\xi \in \mathcal{L}(f_{\mathbf{S}}([b, e]))$. Thus, by Lemma 1, we have that, for every $[b', e'] \sqsubset [b, e]$, $\neg\psi \in \mathcal{L}(f_{\mathbf{S}}([b', e'])$. By the inductive hypothesis, this implies that $\mathbf{M}_{\mathbf{S}}, [b', e'] \Vdash \xi$ for every $[b', e'] \sqsubset [b, e]$, which is a contradiction.

Let v_0 be the root of $\mathbf{S}$. Since $\varphi \in \mathcal{L}(v_0)$ and $f_{\mathbf{S}}([0, 1]) = v_0$, it immediately follows that $\mathbf{M}_{\mathbf{S}}, [0, 1] \Vdash \varphi$. □

4 A Small-Model Theorem for $\mathrm{D}_{\sqsubset}$-Structures

In this section we prove a small-model theorem for $\mathrm{D}_{\sqsubset}$, that is, we will show that a $\mathrm{D}_{\sqsubset}$-formula is satisfiable if and only if there exists a fulfilling $\mathrm{D}_{\sqsubset}$-structure of *bounded size*. To this end, we introduce the following two measurements for $\mathrm{D}_{\sqsubset}$-graph: (i) the *breadth* of a $\mathrm{D}_{\sqsubset}$-graph, which is the maximum number of outgoing edges of a vertex, and (ii) the *depth* of a $\mathrm{D}_{\sqsubset}$-graph, which the maximum length of a *simple path* of vertices (i.e., a path with no repetition of vertices).

We will show that a $\mathrm{D}_{\sqsubset}$-formula is satisfiable if and only if there exists a fulfilling $\mathrm{D}_{\sqsubset}$-structure whose breadth and the depth are linear in the size of the formula. In order to make the proofs easier to understand we provide the following definition.

Definition 7. *Given a* $\mathrm{D}_{\sqsubset}$*-formula* φ *and a* φ*-atom* $A \in \mathcal{A}_{\varphi}$*, the* set of temporal requests of A *is* $\mathrm{REQ}(A) = \{\langle D \rangle \psi \in \mathrm{CL}(\varphi) : \langle D \rangle \psi \in A\}$.

Notice that $\mathrm{REQ}(A)$ identifies all temporal formulae in A: if $\langle D \rangle \psi \notin \mathrm{REQ}(A)$, then $[D]\neg\psi \in A$ (by definition of φ-atom). We denote by REQ_{φ} the set of all $\langle D \rangle$-formulae in $\mathrm{CL}(\varphi)$.

Theorem 3. *For every satisfiable* $\mathrm{D}_{\sqsubset}$*-formula* φ*, there exists a fulfilling* $\mathrm{D}_{\sqsubset}$*-structure whose breadth and depth are bounded by* $2 \cdot |\varphi|$.

Proof. Let $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ be a fulfilling $\mathrm{D}_{\sqsubset}$-structure for φ. The following algorithm builds a fulfilling $\mathrm{D}_{\sqsubset}$-structure $\mathbf{S}' = \langle\langle V', E'\rangle, \mathcal{L}'\rangle$ for φ with the required property.

Initialization. Initialize $\mathbf{S}'$ as the one-vertex $\mathrm{D}_{\sqsubset}$-structure $\langle\langle\{v_0\}, \emptyset\rangle, \mathcal{L}'\rangle$, where v_0 is the root of $\mathbf{S}$ and $\mathcal{L}'(v_0) = \mathcal{L}(v_0)$. Call the procedure *Expansion*(v_0).

Expansion(v). If v is irreflexive, execute *Step 1*; otherwise, execute *Step 2*.

Step 1. Let v' be unique reflexive successor of v in $\mathbf{S}$. Add v' to V' and $(v, v'), (v', v')$ to E'; moreover, put $\mathcal{L}'(v') = \mathcal{L}(v')$. Call *Expansion*$(v')$.

Step 2. Let $\mathrm{REQ}(\mathcal{L}'(v)) = \{\langle D \rangle \psi_1, \ldots, \langle D \rangle \psi_k\}$. Since $\mathbf{S}$ is fulfilling, for every formula $\langle D \rangle \psi_i \in \mathrm{REQ}(\mathcal{L}'(v))$, there exists a descendant v_i of v in $\mathbf{S}$ such that $\psi_i \in \mathcal{L}(v_i)$. For $i = 1 \ldots, k$, add v_i to V' and (v, v_i) to E'; moreover, put $\mathcal{L}'(v_i) = \mathcal{L}(v_i)$. Next, for every v_i such that $\mathrm{REQ}(\mathcal{L}'(v_i)) =$

REQ($\mathcal{L}'(v)$), add an edge (v_i, v) to E'. For the remaining vertices v_i, it holds that REQ($\mathcal{L}'(v_i)$) $\subset$ REQ($\mathcal{L}'(v)$), because every $[D]$-formula in $\mathcal{L}'(v)$ also belongs to $\mathcal{L}'(v_i)$ and there exists at least one ψ_j such that $\langle D\rangle\psi_j \in$ REQ($\mathcal{L}'(v)$) and $[D]\neg\psi_j \in$ REQ($\mathcal{L}'(v_i)$). For every vertex v_i in this latter set, call *Expansion*(v_i).

This algorithm terminates because it calls the *Expansion* procedure on every vertex in V at most once. Moreover, it follows immediately from the construction that it produces a fulfilling $\mathrm{D}_{\sqsubset}$-structure $\mathbf{S}'$ for φ. To prove that both the breadth and the depth of $\mathbf{S}'$ are less than or equal to $2 \cdot |\varphi|$, it suffices to observe that:

- every irreflexive vertex has exactly one outgoing edge;
- the number of outgoing edges of reflexive vertices is bounded by the number of $\langle D\rangle$-formulae in CL(φ), not exceeding the size of the formula;
- in *Step 2*, the procedure *Expansion* is called only on those vertices v_i such that REQ($\mathcal{L}'(v_i)$) $\subset$ REQ($\mathcal{L}'(v)$). It follows that at every step the number of $\langle D\rangle$-formulae strictly decreases. As a consequence, we have that every simple path in $\mathbf{S}'$ contains at most $|\varphi|$ different irreflexive vertices. Since in every simple path reflexive and irreflexive vertices alternate, the depth of the $\mathrm{D}_{\sqsubset}$-structure is bounded by $2 \cdot |\varphi|$. □

Given a formula φ, let n be the number of $\langle D\rangle$-formulae in CL(φ). It follows from Theorem 3 that there is a fulfilling $\mathrm{D}_{\sqsubset}$-structure for φ whose simple vertex paths $v_0, ..., v_k$, starting from the root, have length at most $2n$. Taking advantage of Theorem 1 and Theorem 2, a PSPACE non-deterministic algorithm $\mathrm{D}_{\sqsubset}$-sat for checking the satisfiability of a $\mathrm{D}_{\sqsubset}$-formula φ can be easily obtained as follows. It non-deterministically generates a φ-atom A containing φ and calls a procedure $\mathrm{D}_{\sqsubset}$-step on it. Such a procedure non-deterministically generates a reflexive φ-atom A' such that $A \; D_\varphi \; A'$ and REQ(A') $=$ REQ(A), if there is such atom; otherwise it returns 'NO' and halts. Next, for all $\langle D\rangle\psi \in$ REQ(A'), it non-deterministically generates a φ-atom A'' such that $A' \; D_\varphi \; A''$ and $\psi \in A''$, if there is such atom; otherwise it returns 'NO' and halts. Finally, if REQ(A'') $\neq$ REQ(A'), then it executes a recursive call on A''. $\mathrm{D}_{\sqsubset}$-sat fails and returns 'NO' whenever an atom with the requested properties cannot be generated, and it returns 'YES' if and only if there exists a fulfilling $\mathrm{D}_{\sqsubset}$-structure for φ. It does not exceed polynomial space because the number of nested calls of $\mathrm{D}_{\sqsubset}$-step is bounded by $\mathcal{O}(n)$ (the maximum length of a simple path) and every call needs $\mathcal{O}(n)$ memory space for local operations. In [11] Shapirovsky proved the PSPACE-hardness of $\mathrm{D}_{\sqsubset}$ by providing a reduction of the validity problem for prenex quantified boolean formulae, that is known to be PSPACE-complete, to the satisfiability problem for $\mathrm{D}_{\sqsubset}$.

5 A Tableau Method for $\mathrm{D}_{\sqsubset}$

In the previous section we proved that a $\mathrm{D}_{\sqsubset}$-formula is satisfiable iff it is fulfilled by an effectively constructed $\mathrm{D}_{\sqsubset}$-structure of a suitably bounded size. We will use that construction here to design a sound, complete, and terminating tableau method for $\mathrm{D}_{\sqsubset}$.

A tableau for a $\mathrm{D}_{\sqsubset}$-formula φ is a finite, tree-like graph, in which every node is a subset of $\mathrm{CL}(\varphi)$. Nodes are grouped into *macronodes*, that is, finite subtrees of the tableau, dealt with by the expansion rules. Branching inside a macronode corresponds to disjunctions. Macronodes and edges connecting them represent vertices and edges in the $\mathrm{D}_{\sqsubset}$-structure for φ. We distinguish two types of rules: *local rules*, that generate new nodes belonging to the same macronode, and *global rules*, that generate new nodes belonging to new macronodes.

Given two nodes n and n' such that n' is a descendant of n, we say that n' is a *local descendant* of n (or, equivalently, that n is a *local ancestor* of n') if n and n' belong to the same macronode and that n' is a *global descendant* of n (n is a *global ancestor* of n') if n and n' belong to different macronodes.

Local Rules:

$$\text{(NOT)}\ \frac{\neg\neg\psi, F}{\psi, F} \qquad \text{(OR)}\ \frac{\psi_1 \vee \psi_2, F}{\psi_1, F \mid \psi_2, F} \qquad \text{(AND)}\ \frac{\neg(\psi_1 \vee \psi_2), F}{\neg\psi_1, \neg\psi_2, F}$$

$$\text{(REFL)}\frac{[D]\psi, F}{\psi, [D]\psi, F}$$ where ψ does not occur in any local ancestor of the node

Global Rules:

$$\text{(2-DENS)}\ \frac{[D]\psi_1, \ldots, [D]\psi_m, \langle D\rangle\varphi_1, \ldots, \langle D\rangle\varphi_n, F}{\psi_1, \ldots, \psi_m, [D]\psi_1, \ldots, [D]\psi_m, \langle D\rangle\varphi_1, \ldots, \langle D\rangle\varphi_n}$$

where $m, n \geq 0$ and F contains no temporal formulae;

$$\text{(STEP)}\ \frac{[D]\psi_1, \ldots, [D]\psi_m, \langle D\rangle\varphi_1, \ldots, \langle D\rangle\varphi_n, F}{G_1 \mid \ldots \mid G_n}$$

where $m \geq 0$, $n > 0$, F contains no temporal formulae and, for every $i = 1, \ldots, n$, $G_i = \{\varphi_i, \psi_1, \ldots, \psi_m, [D]\psi_1, \ldots, [D]\psi_m\}$

Reflexive and irreflexive macronodes. As the vertices of a $\mathrm{D}_{\sqsubset}$-graph, macronodes can be either reflexive or irreflexive. A macronode is *irreflexive* if (i) it contains the initial node of the tableau, or (ii) it has been created by an application of the STEP rule. In the other cases, viz., when the macronode has been created by an application of the 2-DENS rule, it is *reflexive.* A node of the tableau is reflexive (resp., irreflexive) if it belongs to a reflexive (resp., irreflexive) macronode.

Expansion strategy. Given a formula φ, the tableau for φ is obtained from the one-node initial tableau $\{\varphi\}$ by recursively applying the following expansion strategy, until it cannot be applied anymore:

1. do not apply the expansion rules to nodes of the tableau containing both ψ and $\neg\psi$, for some $\psi \in \mathrm{CL}(\varphi)$;
2. apply the NOT, OR, and AND rules to both reflexive and irreflexive nodes;

3. apply the 2-DENS rule to irreflexive nodes;
4. apply the REFL and STEP rules to reflexive nodes;

To keep the construction finite, when the application of the 2-DENS rule to a node n would generate a new reflexive node such that there exists another reflexive ancestor node n' in the tableau with the same set of temporal formulae, add an edge from n to n' instead of generating such a new node.

Open and closed tableau. A node n in a tableau $\mathcal{T}$ is *closed* if one of the following conditions holds: (i) there exists a formula $\psi \in \mathrm{CL}(\varphi)$ such that $\psi, \neg\psi \in n$; (ii) in the tableau construction, the NOT, OR, AND, 2-DENS, or REFL rules has been applied to n and *all* the immediate successors of n are closed; (iii) in the tableau construction, the STEP rule has been applied to n and *at least one* of the immediate successors of n is closed.

A tableau $\mathcal{T}$ is *closed* if its initial node is closed; otherwise, it is *open*. Figure 2 depicts a closed tableau for the unsatisfiable formula $\langle D\rangle\langle D\rangle p \wedge \langle D\rangle q \wedge [D]([D]\neg p \vee [D]\neg q)$, with boxes representing macronodes (we underlined formulae to which local rules are applied).

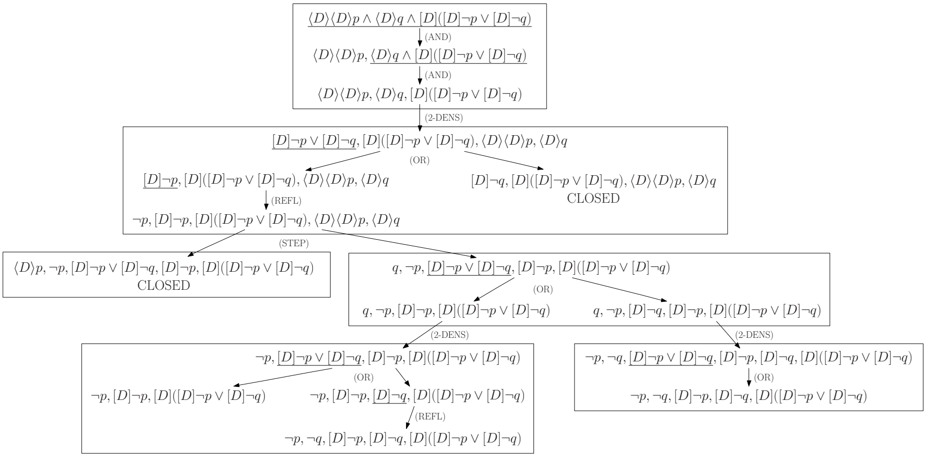

$\underline{\langle D\rangle\langle D\rangle p \wedge \langle D\rangle q \wedge [D]([D]\neg p \vee [D]\neg q)}$
(AND)
$\langle D\rangle\langle D\rangle p, \underline{\langle D\rangle q \wedge [D]([D]\neg p \vee [D]\neg q)}$
(AND)
$\langle D\rangle\langle D\rangle p, \langle D\rangle q, [D]([D]\neg p \vee [D]\neg q)$
(2-DENS)
$\underline{[D]\neg p \vee [D]\neg q}, [D]([D]\neg p \vee [D]\neg q), \langle D\rangle\langle D\rangle p, \langle D\rangle q$
(OR)
$\underline{[D]\neg p}, [D]([D]\neg p \vee [D]\neg q), \langle D\rangle\langle D\rangle p, \langle D\rangle q$
$[D]\neg q, [D]([D]\neg p \vee [D]\neg q), \langle D\rangle\langle D\rangle p, \langle D\rangle q$
CLOSED
(REFL)
$\neg p, [D]\neg p, [D]([D]\neg p \vee [D]\neg q), \langle D\rangle\langle D\rangle p, \langle D\rangle q$
(STEP)
$\langle D\rangle p, \neg p, [D]\neg p \vee [D]\neg q, [D]\neg p, [D]([D]\neg p \vee [D]\neg q)$
CLOSED
$q, \neg p, \underline{[D]\neg p \vee [D]\neg q}, [D]\neg p, [D]([D]\neg p \vee [D]\neg q)$
(OR)
$q, \neg p, [D]\neg p, [D]([D]\neg p \vee [D]\neg q)$
$q, \neg p, [D]\neg q, [D]\neg p, [D]([D]\neg p \vee [D]\neg q)$
(2-DENS)
(2-DENS)
$\neg p, \underline{[D]\neg p \vee [D]\neg q}, [D]\neg p, [D]([D]\neg p \vee [D]\neg q)$
(OR)
$\neg p, [D]\neg p, [D]([D]\neg p \vee [D]\neg q)$
$\neg p, [D]\neg p, \underline{[D]\neg q}, [D]([D]\neg p \vee [D]\neg q)$
(REFL)
$\neg p, \neg q, [D]\neg p, [D]\neg q, [D]([D]\neg p \vee [D]\neg q)$
$\neg p, \neg q, \underline{[D]\neg p \vee [D]\neg q}, [D]\neg p, [D]\neg q, [D]([D]\neg p \vee [D]\neg q)$
(OR)
$\neg p, \neg q, [D]\neg p, [D]\neg q, [D]([D]\neg p \vee [D]\neg q)$

Fig. 2. The tableau for $\langle D\rangle\langle D\rangle p \wedge \langle D\rangle q \wedge [D]([D]\neg p \vee [D]\neg q)$

The proposed tableau method terminates because:

- the local rules can be applied only finitely many times to the nodes of a macronode (the application of the NOT, AND, and OR rules to a node generates one or two nodes where one of the formulae in the original node has been replaced by one or two formulae of strictly lower size; moreover, for any branch in the subtree associated with a macronode and any $[D]$-formula in $\mathrm{CL}(\varphi)$, the REFL rule can be applied at most one time);

- the 2-DENS rule generates a new node only when the set of temporal formulae of such a node is different from that of the other reflexive nodes in the tableau, and thus it can be applied only finitely many times;
- along any branch in the tableau, the applications of the 2-DENS and of STEP rules alternate.

As for the complexity, we have shown that a formula φ is satisfiable if and only if there exists a $\mathrm{D}_{\sqsubset}$-structure for it whose breadth and depth are linear in $|\varphi|$. Such a property holds for any tableau $\mathcal{T}$ for φ as well. Indeed, let n be the number of $\langle D\rangle$-formulae in $\mathrm{CL}(\varphi)$. The number of outgoing edges of a node is bounded by n. Moreover, as in $\mathrm{D}_{\sqsubset}$-structures, every simple path of macronodes starting from the root is of length at most $2n$. Hence, both the breadth and the depth of $\mathcal{T}$ are linear in $|\varphi|$ and a tableau $\mathcal{T}$ for φ can be non-deterministically generated and explored by using a polynomially bounded amount of space. Thus, we obtain the following theorem.

Theorem 4. *The proposed tableau method for* $\mathrm{D}_{\sqsubset}$ *has a PSPACE complexity.*

Theorem 5. *(*SOUNDNESS*) Let φ be a* $\mathrm{D}_{\sqsubset}$*-formula and $\mathcal{T}$ be a tableau for it. If $\mathcal{T}$ is open, then φ is satisfiable.*

Proof. Let $\mathcal{T}$ be an open tableau for φ. We build a fulfilling $\mathrm{D}_{\sqsubset}$-structure $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ for φ step by step, starting from the root of the tableau. Let n_0 be the root of $\mathcal{T}$. Since $\mathcal{T}$ is open, then n_0 is not closed. We generate a one-node $\mathrm{D}_{\sqsubset}$-graph $\langle\{v_0\}, \emptyset\rangle$ and we put the formulae that belong to n_0 in $\mathcal{L}(v_0)$, thus associating n_0 with v_0. Now, let n be a non-closed node in $\mathcal{T}$ and let v be the associated vertex in the $\mathrm{D}_{\sqsubset}$-graph. We distinguish four possible cases, depending on the expansion rule R that has been applied to n in the tableau construction:

Case 1 *R is one of NOT, AND, and REFL.* Since n is not closed, its unique successor n' is not closed as well. We add the formulae contained in n' to $\mathcal{L}(v)$, thus associating n' with v, and then we proceed with the pair (n', v) (note that different nodes can be associated with the same vertex of the $\mathrm{D}_{\sqsubset}$-structure).

- *R is OR.* Since n is not closed, it has a successor n' that is not closed. We add the formulae contained in n' to $\mathcal{L}(v)$, thus associating n' with v, and then we proceed with the pair (n', v).
- *R is 2-DENS.* Since n is not closed, its unique successor n' is not closed as well. If n' has already been associated with a vertex v' during the construction of the $\mathrm{D}_{\sqsubset}$-structure, we simply add an edge (v, v') to E; otherwise, we add a new reflexive vertex v' to V, we add the edges (v, v') and (v', v') to E, and we put the formulae that belong to n' in $\mathcal{L}(v')$, thus associating n' with v'. Then we proceed with the pair (n', v').
- *R is STEP.* Since n is not closed, none of its successors $n_1, \ldots, n_k$ is closed either. Take any of them, n_i. If n_i has already been associated with a vertex v_i during the construction of the $\mathrm{D}_{\sqsubset}$-structure, we simply add an edge (v, v_i) to E; otherwise, we add a new irreflexive vertex v_i to V, we add the edge (v, v_i) to E, and we put the formulae that belong to n_i in $\mathcal{L}(v_i)$, thus associating n_i with v_i. Then we proceed with the pair (n_i, v_i).

Since any tableau for φ is finite, such a construction is terminating. However, the resulting pair $\langle\langle V, E\rangle, \mathcal{L}\rangle$ is not necessarily a $\mathrm{D}_{\sqsubset}$-structure: while $\langle V, E\rangle$ respects the definition of $\mathrm{D}_{\sqsubset}$-graph, the function $\mathcal{L}$ is not necessarily a labeling function. Since in the tableau construction we add new formulae only when necessary, there may exist a vertex $v \in V$ and a formula $\psi \in \mathrm{CL}(\varphi)$ such that neither ψ nor $\neg\psi$ belongs to $\mathcal{L}(v)$. Let $v \in V$ and $\psi \in \mathrm{CL}(\varphi)$ such that neither ψ nor $\neg\psi$ belongs to $\mathcal{L}(v)$. We can complete the definition of $\mathcal{L}$ as follows (by induction on the complexity of ψ):

- if $\psi = p$, with $p \in \mathcal{AP}$, we put $\neg p \in \mathcal{L}(v)$;
- If $\psi = \neg\xi$, we put $\psi \in \mathcal{L}(v)$ if and only if $\xi \notin \mathcal{L}(v)$;
- If $\psi = \psi_1 \vee \psi_2$, we put $\psi_1 \vee \psi_2 \in \mathcal{L}(v)$ if and only if $\psi_1 \in \mathcal{L}(v)$ or $\psi_2 \in \mathcal{L}(v)$;
- If $\psi = \langle D\rangle\xi$, we put $\psi \in \mathcal{L}(v)$ if and only if there exists a descendant v' of v such that $\xi \in \mathcal{L}(v')$.

It follows from the construction that the resulting $\mathrm{D}_{\sqsubset}$-structure $\langle\langle V, E\rangle, \mathcal{L}\rangle$ is a fulfilling $\mathrm{D}_{\sqsubset}$-structure for φ. Therefore, by Theorem 2, φ is satisfiable. □

Theorem 6. (COMPLETENESS) *Let φ be a $\mathrm{D}_{\sqsubset}$-formula and $\mathcal{T}$ be a tableau for it. If $\mathcal{T}$ is closed, then φ is unsatisfiable.*

Proof. Given a node n in a tableau, we say that (the set of formulae belonging to) n is *consistent* if there exists a fulfilling $\mathrm{D}_{\sqsubset}$-structure $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ such that if n belongs to a reflexive (resp., irreflexive) macronode then there exists a reflexive (resp., irreflexive) vertex $v \in V$ such that $\mathcal{L}(v)$ contains all formulae in n; otherwise n is *inconsistent*.

We will prove that for any node n in a tableau $\mathcal{T}$, if n is closed, then n is inconsistent. If there exists a formula $\psi \in \mathrm{CL}(\varphi)$ such that n contains both ψ and $\neg\psi$, then n is obviously inconsistent. In the other cases, we proceed by induction, from the leaves to the root, on the expansion rule R that has been applied to the node n in the construction of the tableau. Since any tableau is finite, we eventually reach the initial node of $\mathcal{T}$, thus concluding that φ is an inconsistent formula.

- *R is NOT.* Then n is of the form $\neg\neg\psi, F$ and it has a unique successor $n' = \psi, F$ within the same macronode. If n' is closed then, by inductive hypothesis, ψ, F is an inconsistent set. Hence, $\neg\neg\psi, F$ is inconsistent.
- *R is OR.* Then n is of the form $\psi_1 \vee \psi_2, F$ and it has two immediate successors $n_1 = \psi_1, F$ and $n_2 = \psi_2, F$ within the same macronode. If both n_1 and n_2 are closed then, by inductive hypothesis, both ψ_1, F and ψ_2, F are inconsistent sets. Hence, $\psi_1 \vee \psi_2, F$ is inconsistent.
- *R is AND.* Then n is of the form $\neg(\psi_1 \vee \psi_2), F$ and it has a unique successor $n' = \neg\psi_1, \neg\psi_2, F$ within the same macronode. If n' is closed then, by inductive hypothesis, $\neg\psi_1, \neg\psi_2, F$ is an inconsistent set. Hence, $\psi_1 \wedge \psi_2, F$ is inconsistent.
- *R is REFL.* In this case, n belongs to a reflexive macronode and it is of the form $[D]\psi, F$. Suppose, for contradiction, that n is closed but consistent,

i.e., that there exists a fulfilling $\mathrm{D}_{\dot\sqsubset}$-structure $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ and a reflexive vertex $v \in V$ such that $n \subseteq \mathcal{L}(v)$. Since v is reflexive, we have that $(v, v) \in E$ and thus $\mathcal{L}(v)\ D_{\varphi}\ \mathcal{L}(v)$. Hence, we have that $\psi \in \mathcal{L}(v)$ and thus the set of formulae $\psi, [D]\psi, F$ is consistent. By inductive hypothesis, this implies that the successor n' of n is not closed, which contradicts the hypothesis that n is closed.

- *R is 2-DENS.* In this case, n belongs to an irreflexive macronode and it is of the form $[D]\psi_1, \ldots, [D]\psi_m, \langle D\rangle\varphi_1, \ldots, \langle D\rangle\varphi_h, F$. Suppose, for contradiction, that n is closed but consistent. Hence, there exists a fulfilling $\mathrm{D}_{\dot\sqsubset}$-structure $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ and an irreflexive vertex $v \in V$ such that $n \subseteq \mathcal{L}(v)$. By the definition of $\mathrm{D}_{\dot\sqsubset}$-structure, v has a reflexive successor v' such that $\mathcal{L}(v)\ D_{\varphi}\ \mathcal{L}(v')$. Hence, v' is such that $\{\psi_1, \ldots, \psi_m, [D]\psi_1, \ldots, [D]\psi_m, \langle D\rangle\varphi_1, \ldots, \langle D\rangle \varphi_h\} \subseteq \mathcal{L}(v')$. This proves that the successor n' of n in the tableau is consistent. By inductive hypothesis, n' is not closed – a contradiction.
- *R is STEP.* In this case, n belongs to a reflexive macronode and it is of the form $[D]\psi_1, \ldots, [D]\psi_m, \langle D\rangle\varphi_1, \ldots, \langle D\rangle\varphi_h, F$. Suppose, for contradiction, that n is closed but consistent. Hence, there exists a fulfilling $\mathrm{D}_{\dot\sqsubset}$-structure $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}\rangle$ and a reflexive vertex $v \in V$ such that $n \subseteq \mathcal{L}(v)$. Since $\mathbf{S}$ is fulfilling, for every formula $\langle D\rangle\varphi_i$ there exists a descendant v_i of v such that $\varphi_i \in \mathcal{L}(v_i)$. This implies that, for every $i = 1, \ldots, h$, the set of formulae $G_i = \{\varphi_i, \psi_1, \ldots, \psi_m, [D]\psi_1, \ldots, [D]\psi_m\}$ is consistent. By inductive hypothesis, this implies that every immediate successor of n is not closed, which contradicts the assumption that n is closed.

Since the tableau $\mathcal{T}$ is closed if and only if its root is closed, and since the root of the tableau is an irreflexive node, from the above result we can conclude that the set $\{\varphi\}$ is inconsistent, namely, that there are no fulfilling $\mathrm{D}_{\dot\sqsubset}$-structures that features an irreflexive vertex v such that $\varphi \in \mathcal{L}(v)$. Since a formula is satisfiable if and only if it belongs to the labelling of the (irreflexive) root of some fulfilling $\mathrm{D}_{\dot\sqsubset}$-structure, this proves that φ is unsatisfiable. □

6 The Logic $\mathrm{D}_{\sqsubset}$ of the Proper Subinterval Relation

For the case of *proper* subinterval relation, we replace $\mathrm{D}_{\dot\sqsubset}$-structures by $\mathrm{D}_{\sqsubset}$-structures. Given an interval $[b, e]$ in a dense linear order $\mathbb{D} = \langle D, <\rangle$, a *beginning subinterval* of $[b, e]$ is any proper subinterval $[b, e']$, for $b < e' < e$; likewise, an *ending subinterval* of $[b, e]$ is any proper subinterval $[b', e]$, for $b < b' < e$.

The presence of the special families of beginning subintervals and ending subintervals of a given interval in a structure with proper subinterval relation causes an essential distinction from the case interval structures with strict subinterval relation studied in the previous sections. This distinction reflects essentially on the respective logic $\mathrm{D}_{\sqsubset}$, and in particular on the definition of $\mathrm{D}_{\sqsubset}$-structures and the construction of tableau for $\mathrm{D}_{\sqsubset}$-formulae, where special care must be taken for these families.

Given a graph $\mathbb{G} = \langle V, E\rangle$, a *maximal cluster* in $\mathbb{G}$ is a maximal set $\mathcal{C}$ of reflexive vertices in V such that for every pair $v, v' \in \mathcal{C}$, v is reachable from v'.

Definition 8. *A finite directed graph $\mathbb{G} = \langle V, E\rangle$ is a* $\mathrm{D}_{\sqsubset}$-graph *if:*

1. *there exists an irreflexive vertex $v_0 \in V$, called the* root *of $\mathbb{G}$, such that any other vertex $v \in V$ is reachable from it;*
2. *every irreflexive vertex $v \in V$ has exactly two successors v_b and v_e, which are both reflexive; we denote by $\mathcal{C}_b$ the maximal cluster containing v_b and by $\mathcal{C}_e$ the maximal cluster containing v_e;*
3. *v_b and v_e have a unique common successor v_c, which is reflexive;*
4. *v_b and v_e may have at most one irreflexive successor and there are no other edges exiting from $\mathcal{C}_b$ and $\mathcal{C}_e$;*
5. *every successor of v_c different from itself is irreflexive.*

A portion of a $\mathrm{D}_{\sqsubset}$-graph is depicted in Figure 3.

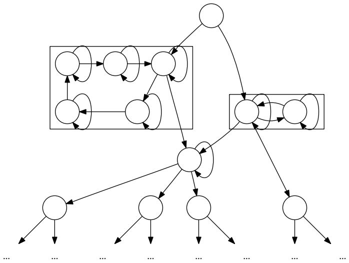


Fig. 3. An example of $\mathrm{D}_{\sqsubset}$-graph

Definition 9. *A* $\mathrm{D}_{\sqsubset}$-structure *is a quadruple $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}, \mathcal{B}, \mathcal{E}\rangle$, where:*

1. *$\langle V, E\rangle$ is a $\mathrm{D}_{\sqsubset}$-graph.*
2. *$\mathcal{L} : V \rightarrow \mathcal{A}_\varphi$ is a* labeling function *that assigns to every vertex $v \in V$ an atom $\mathcal{L}(v)$ such that $\mathcal{L}(v)\ D_\varphi\ \mathcal{L}(v')$ for every edge $(v, v') \in E$.*
3. *$\mathcal{B} : V \rightarrow 2^{\mathrm{REQ}_\varphi}$ and $\mathcal{E} : V \rightarrow 2^{\mathrm{REQ}_\varphi}$ are mappings that assign to every node the set of the requests that must be satisfied respectively in the beginning subintervals and in the ending subintervals of the current interval.*
4. *For every irreflexive vertex $v \in V$, we have that:*
   - *the reflexive vertex v_c is such that $\mathcal{E}(v_c) = \mathcal{B}(v_c) = \emptyset$ and $REQ(\mathcal{L}(v_c)) = REQ(\mathcal{L}(v)) - (\mathcal{B}(v) \cup \mathcal{E}(v))$,*
   - *every reflexive vertex $v' \in \mathcal{C}_b$ is such that $\mathcal{B}(v') = \mathcal{B}(v)$, $\mathcal{E}(v') = \emptyset$ and $REQ(\mathcal{L}(v')) = REQ(\mathcal{L}(v_c)) \cup \mathcal{B}(v)$,*
   - *the unique irreflexive successor v'' from a vertex in $\mathcal{C}_b$ (if any) is such that $\mathcal{B}(v'') \subset \mathcal{B}(v)$ and $\mathrm{REQ}(\mathcal{L}(v'')) \subset \mathrm{REQ}(\mathcal{L}(v))$,*
   - *every reflexive vertex $v' \in \mathcal{C}_e$ is such that $\mathcal{E}(v') = \mathcal{E}(v)$, $\mathcal{B}(v') = \emptyset$ and $REQ(\mathcal{L}(v')) = REQ(\mathcal{L}(v_c)) \cup \mathcal{E}(v)$,*
   - *the unique irreflexive successor v'' from a vertex in $\mathcal{C}_e$ (if any) is such that $\mathcal{E}(v'') \subset \mathcal{E}(v)$ and $\mathrm{REQ}(\mathcal{L}(v'')) \subset \mathrm{REQ}(\mathcal{L}(v))$,*

   where $v_c, \mathcal{C}_b$, and $\mathcal{C}_e$ are defined as in Definition 8.

Let v_0 be the root of $\langle V, E\rangle$. If $\varphi \in \mathcal{L}(v_0)$, we say that $\mathbf{S}$ is a $\mathrm{D}_{\sqsubset}$-*structure for* φ. The notion of fulfilling $\mathrm{D}_{\sqsubset}$-structure is defined as in the case of $\mathrm{D}_{\sqsubset\!\!\cdot}$-structures. An analogous of Theorem 1 holds for $\mathrm{D}_{\sqsubset}$-structures as well.

Theorem 7. *Let* φ *be a* $\mathrm{D}_{\sqsubset}$*-formula which is satisfied in a dense interval model. Then, there exists a fulfilling* $\mathrm{D}_{\sqsubset}$*-structure* $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}, \mathcal{B}, \mathcal{E}\rangle$ *such that* $\varphi \in \mathcal{L}(v_0)$*, where* v_0 *is the root of* $\langle V, E\rangle$*.*

Now, let $\mathbf{S}$ be a fulfilling $\mathrm{D}_{\sqsubset}$-structure for a formula φ. To prove that φ is satisfiable in a dense interval structure we consider the interval $[0, 1]$ of the rational line and define a function $f_{\mathbf{S}}$ mapping intervals in $\mathbb{I}([0, 1])^-$ to vertices in $\mathbf{S}$. Such a function will allow us to define a model for φ.

Definition 10. *Let* $\mathbf{S} = \langle\langle V, E\rangle, \mathcal{L}, \mathcal{B}, \mathcal{E}\rangle$ *be a* $\mathrm{D}_{\sqsubset}$*-structure. The function* $f_{\mathbf{S}} : \mathbb{I}([0, 1])^- \mapsto V$ *is recursively defined as follows. First,* $f_{\mathbf{S}}([0, 1]) = v_0$*. Now, let* $[b, e]$ *be an interval such that* $f_{\mathbf{S}}([b, e]) = v$*, but* $f_{\mathbf{S}}$ *has not been defined for the subintervals of* $[b, e]$*.*

Case 1: v *is an irreflexive vertex. Let* v_b, v_e *be the reflexive successors of* v*,* $\mathcal{C}_b$ *and* $\mathcal{C}_e$ *be their clusters, and* v_c *be the reflexive vertex that they have as common successor. We consider the case when* v_b *has an irreflexive successor* v_b^{max}*,* v_e *has an irreflexive successor* v_e^{max}*, and* v_c *has* k *irreflexive successors* $v_1, \ldots, v_k$*. The other cases are simpler and can be treated in a similar way. Let* $p = \frac{e-b}{2k+3}$*. We define* $f_{\mathbf{S}}$ *in a way similar to the case of* $\mathrm{D}_{\sqsubset\!\!\cdot}$*-structures:*

1. *we put* $f_{\mathbf{S}}([b, b + p]) = v_b^{max}$ *and* $f_{\mathbf{S}}([e - p, e]) = v_e^{max}$*;*
2. *for every* $i = 1, \ldots, k$*, we put* $f_{\mathbf{S}}([b + 2ip, b + (2i + 1)p]) = v_i$*;*
3. *for every* $i = 1, \ldots, k + 1$*, we put* $f_{\mathbf{S}}([b + (2i - 1)p, b + 2ip]) = v_c$*;*
4. *for every* strict *subinterval* $[b', e']$ *of* $[b, e]$ *that is not a subinterval of any of the intervals* $[b + ip, b + (i + 1)p]$*, we put* $f_{\mathbf{S}}([b', e']) = v_c$*.*

To complete the construction we need to define $f_{\mathbf{S}}$ *for the beginning intervals* $[b, e']$ *such that* $b + p < e' < e$ *and for the ending intervals* $[b', e]$ *such that* $b < b' < e - p$*. We map such beginning intervals* $[b, e']$ *to vertices in* $\mathcal{C}_b$ *and ending intervals* $[b', e]$ *to vertices in* $\mathcal{C}_e$*, with the further constraint that for every interval* $[b, e']$ *and every node* $v' \in \mathcal{C}_b$ *there is an interval* $[b, e''] \sqsubset [b, e']$ *such that* $f_{\mathbf{S}}([b, e'']) = v''$ *and, conversely, that for every interval* $[b', e]$ *and every node* $v' \in \mathcal{C}_e$ *there is an interval* $[b'', e] \sqsubset [b'', e]$ *such that* $f_{\mathbf{S}}([b'', e]) = v''$*. The density of the rational line* $[0, 1]$ *allow us to define such a mapping.*

Case 2: v *is a reflexive vertex. We can assume, by construction, that* v *is one of the vertices* v_c *that have only irreflexive successors (except itself). In such a case we proceed as for* $\mathrm{D}_{\sqsubset\!\!\cdot}$*-structures in the definition of* $f_{\mathbf{S}}$*.*

Now, given a fulfilling $\mathrm{D}_{\sqsubset}$-structure $\mathbf{S}$ for φ, we define the corresponding interval model $\mathbf{M}_{\mathbf{S}} = \langle\mathbb{I}([0, 1])^-, \sqsubset, \mathcal{V}\rangle$, where, for every $p \in \mathcal{AP}$, $\mathcal{V}(p) = \{[b, e] : p \in \mathcal{L}(f_{\mathbf{S}}([b, e]))\}$. It turns out that $\mathbf{M}_{\mathbf{S}}$ is a model for φ. The following theorem can be proved by structural induction, similarly to Theorem 2.

Theorem 8. *Given a fulfilling* $\mathrm{D}_{\sqsubset}$*-structure* $\mathbf{S}$ *for* φ*, the corresponding interval model* $\mathbf{M}_{\mathbf{S}} = \langle\mathbb{I}([0, 1])^-, \sqsubset, \mathcal{V}\rangle$ *is such that* $\mathbf{M}_{\mathbf{S}}, [0, 1] \Vdash \varphi$*.*

The following observations imply a bound on the size of $\mathrm{D}_{\sqsubset}$-structures:

- given a cluster of reflexive vertices $\mathcal{C}$, we have that all the vertices in $\mathcal{C}$ share the same set of requests $\mathrm{REQ}_{\mathcal{C}}$. Furthermore, we can remove from $\mathcal{C}$ all the vertices that either does not satisfy any of the $\langle D\rangle$-formulae in $\mathrm{REQ}_{\mathcal{C}}$ or they satisfy only $\langle D\rangle$-formulae that are satisfied by some other vertex in $\mathcal{C}$. This implies that we can build a $\mathrm{D}_{\sqsubset}$-structure where the size of each cluster is bounded by the number of $\langle D\rangle$-formulae in $\mathrm{CL}(\varphi)$.
- we can bound the breadth and depth of the $\mathrm{D}_{\sqsubset\!\!\!\cdot}$-structure by exploiting the same technique that we use for $\mathrm{D}_{\sqsubset}$-structures.

Theorem 9. *For every satisfiable* $\mathrm{D}_{\sqsubset}$*-formula* φ*, there exists a fulfilling* $\mathrm{D}_{\sqsubset}$*-structure with breadth and depth bounded by* $2 \cdot |\varphi|$*.*

As for the complexity, given a formula φ, let n be the number of $\langle D\rangle$-formulae in $\mathrm{CL}(\varphi)$. From the above theorem, every simple path $(v_0, ..., v_n)$ starting from the root of a $\mathrm{D}_{\sqsubset}$-structure for φ has length at most $2n$. A PSPACE non-deterministic algorithm $\mathrm{D}_{\sqsubset}$-sat for checking satisfiability of $\mathrm{D}_{\sqsubset}$ formulae can be obtained by revising the procedure $\mathrm{D}_{\sqsubset\!\!\!\cdot}$-sat as follows. At each step, once it has nondeterministically generated an irreflexive atom, it nondeterministically generates one reflexive atom for v_b, one for v_e, and one for v_c; then, it generates the reflexive atoms for the vertices contained in the cluster of v_b and in the cluster of v_e, if any; finally, it generates the irreflexive successors of v_b and v_e, and one irreflexive successor for v_c, for every $\langle D\rangle$-formula contained in $\mathcal{L}(v_c)$. $\mathrm{D}_{\sqsubset}$-sat fails if an atom with the required properties cannot be generated; otherwise, it succeeds and it generates a fulfilling $\mathrm{D}_{\sqsubset}$-structure for φ. The very same reduction that has been used to prove the PSPACE-hardness of $\mathrm{D}_{\sqsubset\!\!\!\cdot}$ can be applied to $\mathrm{D}_{\sqsubset}$, thus proving that this logic is PSPACE-complete as well.

The tableau method for $\mathrm{D}_{\sqsubset\!\!\!\cdot}$ can be adapted to $\mathrm{D}_{\sqsubset}$, even though such an adaptation is not trivial at all (for lack of space, we omit its description).

7 Conclusions

In this paper we have started building tableau-based decision procedures for logics of subinterval structures. First, we have considered in detail the case of the logic $\mathrm{D}_{\sqsubset\!\!\!\cdot}$; then, we have shown how to refine structures and methods for $\mathrm{D}_{\sqsubset\!\!\!\cdot}$ to cope with the logic $\mathrm{D}_{\sqsubset}$. Our methodology can be suitably adapted to the case we admit the existence of point intervals $[b, b]$ in subinterval structures over dense orderings. The cases of discrete and general orderings present additional challenges which will be treated in subsequent publications.

Acknowledgements

We are grateful to the anonymous referees for useful remarks and constructive criticisms. This work has been funded by the bilateral project "Temporal logics in computer and information sciences", supported by the Italian Ministero degli

Affari Esteri and the National Research Foundation of South Africa, under the Joint Italy/South Africa Science and Technology Agreement. In addition, Davide Bresolin, Angelo Montanari, and Pietro Sala have been supported by the Italian PRIN project on "Constraints and preferences as a unifying formalism for system analysis and solution of real-life problems" and by the European INTAS project on "Algebraic and Deduction Methods in Non Classical Logics and their Applications to Computer Science".

References

1. van Benthem, J.: The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse, Second Edition. Kluver, Norwell (1991)
2. Blackburn, P., de Rijke, M., Venema, V.: Modal Logic. CUP (2001)
3. Bowman, H., Thompson, S.: A decision procedure and complete axiomatization of finite interval temporal logic with projection. Journal of Logic and Computation 13(2), 195–239 (2003)
4. Bresolin, D., Goranko, V., Montanari, A., Sciavicco, G.: On Decidability and Expressiveness of Propositional Interval Neighborhood Logics. In: LFCS 2007. Proc. of the International Symposium on Logical Foundations of Computer Science. LNCS, vol. 4514, pp. 84–99. Springer, Heidelberg (2007)
5. Bresolin, D., Montanari, A.: A tableau-based decision procedure for a branching-time interval temporal logic. In: Schlingloff, H. (ed.) Proc. of the 4th Int. Workshop on Methods for Modalities, pp. 38–53 (2005)
6. Bresolin, D., Montanari, A.: A tableau-based decision procedure for Right Propositional Neighborhood Logic. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS (LNAI), vol. 3702, Springer, Heidelberg (2005)
7. Bresolin, D., Montanari, A., Sala, P.: An optimal tableau-based decision algorithm for Propositional Neighborhood Logic. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, Springer, Heidelberg (2007)
8. Bresolin, D., Montanari, A., Sciavicco, G.: An optimal decision procedure for Right Propositional Neighborhood Logic. Journal of Automated Reasoning 38(1-3), 173–199 (2007)
9. Goranko, V., Montanari, A., Sciavicco, G., Sala, P.: A general tableau method for propositional interval temporal logics: Theory and implementation. Journal of Applied Logic 4(3), 305–330 (2006)
10. Halpem, J.Y., Shoham, Y.: A propositional modal logic of time intervals. Journal of the ACM 38(4), 935–962 (1991)
11. Shapirovsky, I.: On PSPACE-decidability in Transitive Modal Logic. In: Schmidt, R., Pratt-Hartmann, I., Reynolds, M., Wansing, H. (eds.) Advances in Modal Logic, vol. 5, pp. 269–287. King's College Publications, London (2005)
12. Shapirovsky, I., Shehtman, V.: Chronological future modality in Minkowski spacetime. In: Balbiani, P., Suzuki, N.Y., Wolter, F., Zakharyaschev, M. (eds.) Advances in Modal Logic, vol. 4, pp. 437–459. King's College Publications, London (2003)

A Cut-Free Sequent Calculus for Bi-intuitionistic Logic

Linda Buisman[1] and Rajeev Goré[1,2]

[1] The Australian National University
Canberra ACT 0200, Australia
[2] Logic and Computation Programme
Canberra Research Laboratory, NICTA*, Australia
{Linda.Buisman,Rajeev.Gore}@anu.edu.au

Abstract. Bi-intuitionistic logic is the extension of intuitionistic logic with a connective dual to implication. Bi-intuitionistic logic was introduced by Rauszer as a Hilbert calculus with algebraic and Kripke semantics. But her subsequent "cut-free" sequent calculus for `BiInt` has recently been shown by Uustalu to fail cut-elimination. We present a new cut-free sequent calculus for `BiInt`, and prove it sound and complete with respect to its Kripke semantics. Ensuring completeness is complicated by the interaction between implication and its dual, similarly to future and past modalities in tense logic. Our calculus handles this interaction using extended sequents which pass information from premises to conclusions using variables instantiated at the leaves of failed derivation trees. Our simple termination argument allows our calculus to be used for automated deduction, although this is not its main purpose.

1 Introduction

Propositional intuitionistic logic (`Int`) has connectives $\rightarrow$, $\wedge$, $\vee$ and $\neg$, with $\neg\varphi$ definable as $\neg\varphi := \varphi \rightarrow\bot$. Propositional dual intuitionistic logic (`DualInt`) has connectives $-\!\!<$, $\wedge$, $\vee$ and $\sim$, with $\sim\varphi$ definable as $\sim\varphi := \top -\!\!< \varphi$. Bi-intuitionistic logic (`BiInt`) or subtractive logic or Heyting-Brouwer logic is the union of `Int` and `DualInt`. It is a conservative extension of both and was first studied by Rauszer [11,12].

Rauszer's Kripke semantics for `BiInt` involve a reflexive and transitive binary relation $\mathcal{R}$, and its converse $\mathcal{R}^{-1}$, similar to the normal **tense** logic `Kt.S4`. Specifically, a world w makes $\varphi \rightarrow \psi$ true if every $\mathcal{R}$-successor v that makes φ true also makes ψ true, and a world w makes $\varphi -\!\!< \psi$ true if there exists an $\mathcal{R}$-predecessor v where φ holds but ψ does not. Thus, $\varphi -\!\!< \psi$ ("φ *excludes* ψ") is a natural dual to $\varphi \rightarrow \psi$ ("φ *implies* ψ").

While there are many cut-free sequent systems for `Int` (e.g., [15,6,5]) and `DualInt` (e.g., [16,4]), the case for `BiInt` is less satisfactory. Rauszer presented

* National ICT Australia is funded by the Australian Government's Dept of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence program.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 90–106, 2007.

a sequent calculus for BiInt in [11] and "proved" it cut-free, but Uustalu [17] has recently shown that the BiInt-valid formula $p \to (q \vee (r \to ((p \multimap\!\!< q) \wedge r))$ cannot be derived in Rauszer's calculus without the cut rule. Uustalu's example also shows that Crolard's sequent calculus [3] for BiInt is not cut-free. Uustalu's example fails in these calculi because certain sequent rules are restricted to singleton succedents or antecedents in their conclusions, and these fail to capture the interaction between $\to$ and $-\!\!<$. Uustalu and Pinto have apparently given a cut-free sequent-calculus for BiInt [19,18] using labelled formulae which use the Kripke semantics directly in the rules. But we have been unable to examine their rules or proofs, as only the abstract of their work has been published.

We present a new purely syntactic cut-free sequent calculus for BiInt. We avoid Rauszer's and Crolard's restrictions on the antecedents and succedents for certain rules by basing our rules on Dragalin's GHPC [5] which allows multiple formulae on both sides of sequents. To maintain intuitionistic soundness, we restrict the *premise* of the implication-right rule to a singleton in the succedent. Dually, the premise of our exclusion-left rule is restricted to a singleton in the antecedent. But using Dragalin's calculus and its dual does not give us BiInt completeness. We therefore follow Schwendimann [13], and use sequents which pass relevant information from premises to conclusions using variables instantiated at the leaves of failed derivation trees. We then recompute parts of our derivation trees using the new information, similarly to the restart technique of [9]. Our calculus thus uses a purely syntactic addition to traditional sequents, rather than resorting to a semantic mechanism such as labels. Our termination argument also relies on two new rules from Śvejdar [14].

If we were interested only in decision procedures, we could obtain a decision procedure for BiInt by embedding it into the tense logic Kt.S4 [20], and using tableaux for description logics with inverse roles [9]. However, an embedding into Kt.S4 provides no proof-theoretic insights into BiInt itself. Moreover, the restart technique of Horrocks et al. [9] involves non-deterministic expansion of disjunctions, which is complicated by inverse roles. Their actual implementation avoids this non-determinism by keeping a global view of the whole counter-model under construction. In contrast, we handle this non-determinism by syntactically encoding it using variables and extended formulae, neither of which have a semantic content. Our purely syntactic approach is preferable for proof-theoretic reasons, since models are never explicitly involved in the proof system: see Remark 2.

In Section 2, we define the syntax and semantics of BiInt. In Section 3, we introduce our sequent calculus **GBiInt** and give an example derivation of Uustalu's interaction formula. We prove the soundness and completeness of **GBiInt** in Sections 4 and 5 respectively. A version with full proofs can be found in [2].

2 Syntax and Semantics of BiInt

The formulae *Fml* of BiInt are built from a denumerable set of *Atoms* and the constants $\top$ and $\bot$ using the connectives $\wedge$, $\vee$, $\to$, $-\!\!<$, $\neg$, and $\sim$. The length of a formula χ is just the number of symbols it contains. We use classical first-order

$$\begin{array}{lll}
w \vDash \varphi \vee \psi & \text{if} & w \vDash \varphi \text{ or } w \vDash \psi \\
w \vDash \varphi \wedge \psi & \text{if} & w \vDash \varphi \ \& \ w \vDash \psi \\
w \vDash \neg\varphi & \text{if} & \forall u \in \mathcal{W}.[w\mathcal{R}u \Rightarrow (u \nvDash \varphi)] \\
w \vDash \varphi \rightarrow \psi & \text{if} & \forall u \in \mathcal{W}.[w\mathcal{R}u \Rightarrow (u \nvDash \varphi \text{ or } u \vDash \psi)] \\
w \vDash \sim\varphi & \text{if} & \exists u \in \mathcal{W}.[u\mathcal{R}w \ \& \ u \nvDash \varphi] \\
w \vDash \varphi \prec \psi & \text{if} & \exists u \in \mathcal{W}.[u\mathcal{R}w \ \& \ u \vDash \varphi \ \& \ u \nvDash \psi]
\end{array}$$

Fig. 1. `BiInt` semantics

logic when reasoning about `BiInt` at the meta-level. A `BiInt` frame is a pair $\langle\mathcal{W}, \mathcal{R}\rangle$, where $\mathcal{W}$ is a non-empty set of worlds and $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{W}$ is a binary reflexive transitive relation. A `BiInt` model is a triple $\mathcal{M} = \langle\mathcal{W}, \mathcal{R}, \vartheta\rangle$, where $\langle\mathcal{W}, \mathcal{R}\rangle$ is a `BiInt` frame and the truth valuation ϑ is a function $\mathcal{W} \times Atoms \rightarrow \{\text{true}, \text{false}\}$ which obeys: $\forall w \in \mathcal{W}.\vartheta(w, \top) = \text{true}$; $\forall w \in \mathcal{W}.\vartheta(w, \bot) = \text{false}$; and which obeys persistence, also known as truth monotonicity:

$$\forall u, w \in \mathcal{W}.\forall p \in Atoms.(\vartheta(w, p) = \text{true} \ \& \ w\mathcal{R}u) \Rightarrow (\vartheta(u, p) = \text{true}).$$

Given a model $\mathcal{M} = \langle\mathcal{W}, \mathcal{R}, \vartheta\rangle$, a world $w \in \mathcal{W}$ and an atom $p \in Atoms$, we write $w \vDash p$ if $\vartheta(w, p) = \text{true}$. We pronounce $\vDash$ as "forces", and we pronounce $\nvDash$ as "rejects". The forcing of compound formulae is defined in Fig. 1. Since $\neg$ and $\sim$ can be derived from $\rightarrow$ and $\prec$ respectively, we restrict our attention to $\rightarrow$, $\prec$, $\wedge$, $\vee$. We obtain persistence for compound formulae by induction on their length, and then reverse persistence for compound formulae follows from persistence because the truth valuation is binary:

$$\begin{array}{l}
\forall\mathcal{M} = \langle\mathcal{W}, \mathcal{R}, \vartheta\rangle.\forall u, w \in \mathcal{W}.\forall\varphi \in Fml.(w \vDash \varphi \ \& \ w\mathcal{R}u \Rightarrow u \vDash \varphi) \\
\forall\mathcal{M} = \langle\mathcal{W}, \mathcal{R}, \vartheta\rangle.\forall u, w \in \mathcal{W}.\forall\varphi \in Fml.(w \nvDash \varphi \ \& \ u\mathcal{R}w \Rightarrow u \nvDash \varphi).
\end{array}$$

We write ϵ to mean the empty set. Given a formula φ and two sets of formulae Δ and Γ, we write Δ, Γ for $\Delta \cup \Gamma$ and we write Δ, φ for $\Delta \cup \{\varphi\}$. Given a model $\mathcal{M} = \langle\mathcal{W}, \mathcal{R}, \vartheta\rangle$ and a world $w \in \mathcal{W}$, we write $w \vDash \Gamma$ (w forces Γ) if $\forall\varphi \in \Gamma.w \vDash \varphi$, and we write $w \dashv \Delta$ (w rejects Δ) if $\forall\varphi \in \Delta.w \nvDash \varphi$. We deliberately use "$\dashv$" for rejection of sets to emphasize that every member of the set is rejected, instead of "$\nvDash$", which could be seen as "some member is rejected".

$$\begin{array}{l}
\Gamma \Vdash_{\texttt{BiInt}} \Delta \text{ means: } \forall\mathcal{M} = \langle\mathcal{W}, \mathcal{R}, \vartheta\rangle.\forall w \in \mathcal{W}.(w \vDash \Gamma \Rightarrow \exists\varphi \in \Delta.w \vDash \varphi) \\
\Gamma \nVdash_{\texttt{BiInt}} \Delta \text{ means: } \exists\mathcal{M} = \langle\mathcal{W}, \mathcal{R}, \vartheta\rangle.\exists w \in \mathcal{W}.(w \vDash \Gamma \ \& \ w \dashv \Delta).
\end{array}$$

Thus $\Gamma \nVdash_{\texttt{BiInt}} \Delta$ means that $\Gamma \Vdash_{\texttt{BiInt}} \Delta$ is falsifiable. As usual, our sequent calculus has a semantic reading which assumes that there exists an initial world w_0 in a `BiInt`-model $\mathcal{M}$ where $w_0 \vDash \Gamma$ and $w_0 \dashv \Delta$. We then systematically apply the sequent rules using backward proof-search to either construct $\mathcal{M}$ successfully, giving us $\Gamma \nVdash_{\texttt{BiInt}} \Delta$, or conclude that $\mathcal{M}$ cannot exist, giving us $\Gamma \Vdash_{\texttt{BiInt}} \Delta$.

3 Our Sequent Calculus GBiInt

We now present **GBiInt**, a Gentzen-style sequent calculus for `BiInt`. The sequents have a non-traditional component in the form of variables that are instantiated at the leaves of the derivation tree, and passed back to lower sequents from *premises to conclusion.* Note that variables are not names for Kripke worlds.

We extend our syntax for presenting some of our sequent rules. The extended `BiInt` formulae are defined as: if φ is a `BiInt` formula, then φ is an extended `BiInt` formula, and if $\mathcal{S}/\mathcal{P}$ is a set $\{\{\varphi_0^0, \cdots, \varphi_0^n\}, \cdots, \{\varphi_m^0, \cdots, \varphi_m^k\}\}$ of sets of `BiInt` formulae, then $\bigvee \mathcal{S}$ and $\bigwedge \mathcal{P}$ are extended `BiInt` formulae with intended semantics

$$\bigvee \mathcal{S} \equiv (\varphi_0^0 \wedge \cdots \wedge \varphi_0^n) \vee \cdots \vee (\varphi_m^0 \wedge \cdots \wedge \varphi_m^k)$$
$$\bigwedge \mathcal{P} \equiv (\varphi_0^0 \vee \cdots \vee \varphi_0^n) \wedge \cdots \wedge (\varphi_m^0 \vee \cdots \vee \varphi_m^k).$$

From now on, we implicitly treat extended `BiInt` formulae as their `BiInt` equivalents. Given a `BiInt` model $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \vartheta \rangle$, and a world $w \in \mathcal{W}$, the following semantics follows directly from their definition:

$$w \vDash \bigvee \mathcal{S} \text{ if } \exists \Gamma \in \mathcal{S}. w \vDash \Gamma \quad \text{and} \quad w =\!\mid \bigwedge \mathcal{P} \text{ if } \exists \Delta \in \mathcal{P}. w =\!\mid \Delta.$$

We can now extend the definition of forcing and rejecting to extended `BiInt` formulae in the obvious way. If Γ and Δ are sets of extended `BiInt` formulae, and φ is an extended `BiInt` formula, then $w \vDash \Gamma$ if $\forall \varphi \in \Gamma. w \vDash \varphi$, and $w =\!\mid \Delta$ if $\forall \varphi \in \Delta. w \nvDash \varphi$.

A **GBiInt** sequent is an expression $\frac{\mathcal{S}}{\mathcal{P}} || \, \Gamma \vdash \Delta$, where the left hand side (LHS) Γ is a set of extended `BiInt` formulae; the right hand side (RHS) Δ is a set of extended `BiInt` formulae; and the variables $\mathcal{S}$ and $\mathcal{P}$ are each a set of sets of formulae. We sometimes write just $\Gamma \vdash \Delta$, ignoring the variable values for readability, but only when the values of the variables are not important to the discussion. In terms of the counter-model under construction, we say that a sequent $\frac{\mathcal{S}}{\mathcal{P}} || \, \Gamma \vdash \Delta$ is **falsifiable** [at w_0 in $\mathcal{M}$] iff there exists a `BiInt` model $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \vartheta \rangle$ and $\exists w_0 \in \mathcal{W}$ such that $w_0 \vDash \Gamma$ and $w_0 =\!\mid \Delta$. Thus, a sequent $\Gamma \vdash \Delta$ is not falsifiable iff $\Gamma \Vdash_{\texttt{BiInt}} \Delta$. We say the **variable conditions** of a sequent $\gamma = \frac{\mathcal{S}}{\mathcal{P}} || \, \Gamma \vdash \Delta$ hold iff γ is falsifiable at w_0 in some model $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \vartheta \rangle$ and the following **S**uccessor/**P**redecessor conditions hold:

$\mathcal{S}$-condition: $\exists \Sigma \in \mathcal{S}. \forall w \in \mathcal{W}. w_0 \mathcal{R} w \Rightarrow w \vDash \Sigma$
$\mathcal{P}$-condition: $\exists \Pi \in \mathcal{P}. \forall w \in \mathcal{W}. w \mathcal{R} w_0 \Rightarrow w =\!\mid \Pi.$

A sequent **rule** is an expression of one of the two forms below

$$(name) \frac{\gamma_1 \ \cdots \ \gamma_n}{\gamma} \qquad\qquad (name) \; \frac{\gamma_0 \ \cdots \ \gamma_n}{\gamma}$$
$$\text{side conditions} \qquad\qquad\qquad \text{side conditions}$$

where $n \geq 0$, and each γ_i is a sequent. The rule has a name, a **conclusion** γ, optional **premise(s)** $\gamma_1, \cdots, \gamma_n$, optional **side conditions**, and universal branching as indicated by a solid line or existential branching as indicated by a dashed line (explained shortly).

Our **traditional** rules (Fig. 2) are based on Dragalin's `GHPC` [5] for `Int` because we require multiple formulae in the succedents and antecedents of sequents for completeness; we have added symmetric rules for the `DualInt` connective $\prec$.

$$(Id)\ \frac{}{\substack{\mathcal{S}:=\epsilon\\ \mathcal{P}:=\epsilon}\Big|\Big|\ \Gamma, \varphi \vdash \Delta, \varphi} \qquad (\bot_L)\ \frac{}{\substack{\mathcal{S}:=\epsilon\\ \mathcal{P}:=\epsilon}\Big|\Big|\ \Gamma, \bot \vdash \Delta} \qquad (\top_R)\ \frac{}{\substack{\mathcal{S}:=\epsilon\\ \mathcal{P}:=\epsilon}\Big|\Big|\ \Gamma \vdash \Delta, \top}$$

$$(\wedge_L)\ \frac{\substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma, \varphi \wedge \psi, \varphi, \psi \vdash \Delta}{\substack{\mathcal{S}:=\mathcal{S}_1\\ \mathcal{P}:=\mathcal{P}_1}\Big|\Big|\ \Gamma, \varphi \wedge \psi \vdash \Delta} \qquad (\wedge_R)\ \frac{\substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma \vdash \Delta, \varphi \wedge \psi, \varphi \qquad \substack{\mathcal{S}_2\\ \mathcal{P}_2}\Big|\Big|\ \Gamma \vdash \Delta, \varphi \wedge \psi, \psi}{\substack{\mathcal{S}:=\mathcal{S}_1 \cup \mathcal{S}_2\\ \mathcal{P}:=\mathcal{P}_1 \cup \mathcal{P}_2}\Big|\Big|\ \Gamma \vdash \Delta, \varphi \wedge \psi}$$

$$(\vee_R)\ \frac{\substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma \vdash \Delta, \varphi \vee \psi, \varphi, \psi}{\substack{\mathcal{S}:=\mathcal{S}_1\\ \mathcal{P}:=\mathcal{P}_1}\Big|\Big|\ \Gamma \vdash \Delta, \varphi \vee \psi} \qquad (\vee_L)\ \frac{\substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma, \varphi \vee \psi, \varphi \vdash \Delta \qquad \substack{\mathcal{S}_2\\ \mathcal{P}_2}\Big|\Big|\ \Gamma, \varphi \vee \psi, \psi \vdash \Delta}{\substack{\mathcal{S}:=\mathcal{S}_1 \cup \mathcal{S}_2\\ \mathcal{P}:=\mathcal{P}_1 \cup \mathcal{P}_2}\Big|\Big|\ \Gamma, \varphi \vee \psi \vdash \Delta}$$

$$(\rightarrow_L)\ \frac{\substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma, \varphi \rightarrow \psi \vdash \varphi, \Delta \qquad \substack{\mathcal{S}_2\\ \mathcal{P}_2}\Big|\Big|\ \Gamma, \varphi \rightarrow \psi, \psi \vdash \Delta}{\substack{\mathcal{S}:=\mathcal{S}_1 \cup \mathcal{S}_2\\ \mathcal{P}:=\mathcal{P}_1 \cup \mathcal{P}_2}\Big|\Big|\ \Gamma, \varphi \rightarrow \psi \vdash \Delta}$$

$$(-\!\!<_R)\ \frac{\substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma, \psi \vdash \Delta, \varphi -\!\!< \psi \qquad \substack{\mathcal{S}_2\\ \mathcal{P}_2}\Big|\Big|\ \Gamma \vdash \Delta, \varphi -\!\!< \psi, \varphi}{\substack{\mathcal{S}:=\mathcal{S}_1 \cup \mathcal{S}_2\\ \mathcal{P}:=\mathcal{P}_1 \cup \mathcal{P}_2}\Big|\Big|\ \Gamma \vdash \Delta, \varphi -\!\!< \psi}$$

For every rule with premises π_i and conlusion γ, apply the rule only if:
$\forall \pi_i.(LHS_{\pi_i} \not\subseteq LHS_\gamma$ or $RHS_{\pi_i} \not\subseteq RHS_\gamma)$

Fig. 2. GBiInt rules - traditional

The main difference is that our $(\rightarrow_L)$ rule and the symmetric $(-\!\!<_R)$ carry their principal formula and all side formulae into the premises. Our rules for $\wedge$ and $\vee$ also carry their principal formula into their premises to assist with termination. Note that there are other approaches to a terminating sequent calculus for `Int`, e.g., Dyckhoff's contraction-free calculi [6], or history methods by Heuerding et al. [8] and Howe [10]. These methods are less suitable when the interaction between `Int` and `DualInt` formulae needs to be considered, since they erase potentially relevant formulae too soon during backward proof search. Moreover, we found it easier to prove semantic completeness with our loop-checking method than with history-based methods since both [8] and [10] prove completeness using syntactic transformations of derivations. Consequently, while **GBiInt** is sound and complete for the `Int` (and `DualInt`) fragment of `BiInt`, it is unlikely to be as efficient on the fragment as these specific calculi.

Our rules for $\rightarrow$ on the right and $-\!\!<$ on the left (Fig. 3) are **non-traditional**. The $(\rightarrow_R)$ and $(-\!\!<_L)$ rules have *two* premises instead of one, and they are connected by **existential branching** as indicated by the dotted horizontal line. Existential branching means that the conclusion is derivable if *some* premise is derivable; thus it is dual to the conventional universal branching, where the conclusion is derivable if *all* premises are derivable. We chose existential branching rather than two separate non-invertible rules so the left premise can communicate information via variables to the right premise. This inter-premise communication and the use of variables is crucial to proving interaction formulae of `BiInt`, and it gives our calculus an **operational reading**.

When applying an existential branching rule during backward proof search, we first create the left premise. If the left premise is non-derivable, then it returns

$$(Ret)\ \frac{}{\substack{\mathcal{S}:=\{\Gamma\}\\ \mathcal{P}:=\{\Delta\}}\Big|\Big|\ \Gamma \vdash \Delta}$$

where no other rule is applicable

$$(\to^I_R)\ \frac{\substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma \vdash \Delta, \varphi \to \psi, \psi}{\substack{\mathcal{S}:=\mathcal{S}_1\\ \mathcal{P}:=\mathcal{P}_1}\Big|\Big|\ \Gamma \vdash \Delta, \varphi \to \psi} \qquad (-\!\!<^I_L)\ \frac{\substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma, \varphi, \varphi -\!\!< \psi \vdash \Delta}{\substack{\mathcal{S}:=\mathcal{S}_1\\ \mathcal{P}:=\mathcal{P}_1}\Big|\Big|\ \Gamma, \varphi -\!\!< \psi \vdash \Delta}$$

$$(\to_R)\ \begin{array}{c} \substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma, \varphi \vdash \psi \qquad \substack{\mathcal{S}_2\\ \mathcal{P}_2}\Big|\Big|\ \Gamma \vdash \Delta, \varphi \to \psi, \bigwedge \mathcal{P}_1 \\ \hdashline \mathcal{S}/\mathcal{P}:=\begin{cases} \mathcal{S}_1/\mathcal{P}_1 & \text{if } \mathcal{P}_1 = \epsilon \\ \mathcal{S}_2/\mathcal{P}_2 & \text{if right prem created} \\ \{\Gamma\}/\{\Delta, \varphi \to \psi\} & \text{otherwise} \end{cases} \Bigg|\Bigg|\ \Gamma \vdash \Delta, \varphi \to \psi \\ \text{right prem created only if } \mathcal{P}_1 \neq \epsilon \ \&\ \forall \Pi_i \in \mathcal{P}_1 . \Pi_i \not\subseteq \{\Delta, \varphi \to \psi\} \end{array}$$

$$(-\!\!<_L)\ \begin{array}{c} \substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \varphi \vdash \Delta, \psi \qquad \substack{\mathcal{S}_2\\ \mathcal{P}_2}\Big|\Big|\ \Gamma, \varphi -\!\!< \psi, \bigvee \mathcal{S}_1 \vdash \Delta \\ \hdashline \mathcal{S}/\mathcal{P}:=\begin{cases} \mathcal{S}_1/\mathcal{P}_1 & \text{if } \mathcal{S}_1 = \epsilon \\ \mathcal{S}_2/\mathcal{P}_2 & \text{if right prem created} \\ \{\Gamma, \varphi -\!\!< \psi\}/\{\Delta\} & \text{otherwise} \end{cases} \Bigg|\Bigg|\ \Gamma, \varphi -\!\!< \psi \vdash \Delta \\ \text{right prem created only if } \mathcal{S}_1 \neq \epsilon \ \&\ \forall \Sigma_i \in \mathcal{S}_1 . \Sigma_i \not\subseteq \{\Gamma, \varphi -\!\!< \psi\} \end{array}$$

$$(\bigwedge_R)\ \frac{\substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma \vdash \Delta, \Pi_1 \ \cdots\ \substack{\mathcal{S}_n\\ \mathcal{P}_n}\Big|\Big|\ \Gamma \vdash \Delta, \Pi_n}{\substack{\mathcal{S}:=\bigcup_1^n \mathcal{S}_i\\ \mathcal{P}:=\bigcup_1^n \mathcal{P}_i}\Big|\Big|\ \Gamma \vdash \Delta, \bigwedge \Pi} \qquad (\bigvee_L)\ \frac{\substack{\mathcal{S}_1\\ \mathcal{P}_1}\Big|\Big|\ \Gamma, \Sigma_1 \vdash \Delta \ \cdots\ \substack{\mathcal{S}_n\\ \mathcal{P}_n}\Big|\Big|\ \Gamma, \Sigma_n \vdash \Delta}{\substack{\mathcal{S}:=\bigcup_1^n \mathcal{S}_i\\ \mathcal{P}:=\bigcup_1^n \mathcal{P}_i}\Big|\Big|\ \Gamma, \bigvee \Sigma \vdash \Delta}$$

For every universally branching rule with premises π_i and conlusion γ,
apply the rule only if: $\forall \pi_i.(LHS_{\pi_i} \not\subseteq LHS_\gamma$ or $RHS_{\pi_i} \not\subseteq RHS_\gamma)$

For every existentially branching rule with left premise π and conlusion γ,
apply the rule only if: $LHS_\pi \not\subseteq LHS_\gamma$ or $RHS_\pi \not\subseteq RHS_\gamma$

Fig. 3. GBiInt rules - non-traditional

the variables $\mathcal{S}_1$ and $\mathcal{P}_1$. We then use these variables to create the right premise, which corresponds to the same world as the conclusion, but with updated information. Our existential branching rules work together with (*Ret*), which assigns the variables at non-derivable leaves of failed derivation trees, and $(\bigwedge_R)$ and $(\bigvee_L)$, which extract the different variable choices at existential branching rules.

The conclusion of each of our rules **assigns the variables** based on the variables returned from the premise(s), and we use the indices $i, 1, 2$ to indicate the premise from which the variable takes its value. For rules with a single premise, the variables are simply passed down from premise to conclusion. For example, the conclusion of $(\wedge_L)$ in Fig. 2 assigns $\mathcal{S} := \mathcal{S}_1$, where $\mathcal{S}_1$ is the value of the variable at the premise. However, for rules with multiple universally branching premises, we take a union of the sets of sets corresponding to each falsifiable premise. For example, the conclusion of $(\bigwedge_R)$ in Fig. 3 assigns $\mathcal{S} := \bigcup_1^n \mathcal{S}_i$, where $\mathcal{S}_i$ is the value of the variable at the i-th premise.

This way, the sets of sets stored in our variables **determinise** the return of formulae to lower sequents – each non-derivable premise corresponds to an open branch, and at this point we do not know whether it will stay open once processed in conjunction with lower sequents. Therefore, we need to temporarily keep all open branches: see Example 2 in [2]. Then the intuition behind adding $\bigwedge \mathcal{P}$ to the right premise of $(\rightarrow_R)$ is that the subsequent application of $(\bigwedge_R)$ will create one or more premises, depending on the cardinality of $\mathcal{P}$. Since $\mathcal{P}$ is a set of sets representing all the open branches, all of the premises of $(\bigwedge_R)$ have to be derivable in order to obtain a derivation. On the other hand, if some premises of $(\bigwedge_R)$ are non-derivable (open), we form the set that consists of the union of the variables returned by those premises, and pass the union back to lower sequents, and so on. The premises that are derivable contribute only ϵ and are thus ignored by the union operator. Also, we only create the right premise of $(\rightarrow_R)$ if **every** member of $\mathcal{P}$ introduces new formulae to the current world. Otherwise, the current world already contains one of the open branches, which would still remain open after an application of $(\bigwedge_R)$. To summarise, the sets-of-sets concept of variables is critical to the soundness of **GBiInt**, as it allows us to remember the required choices arising further up the tree.

The **extended syntax** allows us to syntactically encode the variable choices described above. While the variables $\mathcal{S}$ and $\mathcal{P}$ are sets of sets when we pass them down the tree and combine them using set union, we use $\bigvee \mathcal{S}$ on the left and $\bigwedge \mathcal{P}$ on the right of the sequent to reflect these choices when we add $\bigvee \mathcal{S}$ or $\bigwedge \mathcal{P}$ to the right premise of an existentially branching rule. Then the $(\bigvee_L)$ and $(\bigwedge_R)$ rules break down the extended formulae $\bigvee \mathcal{S}$ and $\bigwedge \mathcal{P}$ to yield several premises, each corresponding to one variable choice. Thus the extended syntax allows us to give an intuitive syntactic representation of the variable choices.

We have also added the rule $(\rightarrow_R^I)$ for implication on the right (and dually, $(\prec_L^I)$) originally given by Śvejdar [14]. Rather than immediately creating the successor for a rejected $\varphi \rightarrow \psi$, the $(\rightarrow_R^I)$ rule first pre-emptively adds ψ to the right hand side of the sequent. Although Śvejdar himself does not give the semantics behind this rule, and is unable to explain the precise role it plays in his calculus, it is very useful in our termination proof. The rule effectively uses the reverse persistence property – if some successor v forces φ and rejects ψ, then the current world w must reject ψ too, for if w forces ψ, then by forward persistence so does v, thus giving a contradiction.

The **side condition** on each of our rules is a general **blocking** condition, where we only explore the premise(s), if they are different from the conclusion. For example, in the $(\wedge_R)$ case, the blocking condition means that we apply the rule in backward proof search only if $\varphi \notin \Delta$ and $\psi \notin \Delta$, since otherwise some premise would be equal to the conclusion.

GBiInt also has the **subformula property**. This is obvious for all rules, except $(\rightarrow_R)$ and the dual $(\prec_L)$. For these, the right premise "constructs" the formulae $\bigwedge \mathcal{P}$ and $\bigvee \mathcal{S}$. However, since $\mathcal{P}$ and $\mathcal{S}$ are sets of sets of subformulae of the conclusion that are again extracted by $(\bigwedge_R)$ and $(\bigvee_L)$, the right premise of $(\rightarrow_R)$ and $(\prec_L)$ effectively only contains subformulae of the conclusion.

A **GBiInt** tree for ${}^{\mathcal{S}}_{\mathcal{P}}\big|\big|\, \Gamma \vdash \Delta$ is a tree rooted at ${}^{\mathcal{S}}_{\mathcal{P}}\big|\big|\, \Gamma \vdash \Delta$ where each child is obtained by a backwards application of a **GBiInt** rule and each leaf is an instance of $(\bot_L)$, $(\top_R)$, (Id) or (Ret).

Definition 1. *A* **GBiInt** *tree for* $\gamma = {}^{\mathcal{S}}_{\mathcal{P}}\big|\big|\, \Gamma \vdash \Delta$ *is a derivation if:* γ *is the conclusion of a* $(\bot_L)$*,* $(\top_R)$ *or* (Id) *rule application; OR* γ *is the conclusion of a* ***universal branching*** *rule application and* ***all*** *its* ***premises*** *are derivations; OR* γ *is the conclusion of an* ***existential branching*** *rule application and* ***some premise*** *is a derivation.*

In the following example, we use a simplified version of the $(\wedge_R)$ rule, which discards the principal formula from the premises, merely to save horizontal space. Also, we only show non-empty variable values.

Example 1. The following is a derivation tree of Uustalu's interaction formula $p \to (q \vee (r \to ((p -\!\!< q) \wedge r))$, simplified to the sequent $p \vdash q, r \to ((p -\!\!< q) \wedge r)$. Let $X := r \to ((p -\!\!< q) \wedge r)$. The tree should be read bottom-up while ignoring the variables $\mathcal{S}$ and $\mathcal{P}$. At the leaves, the variables are assigned and transmit information down to parents and across to some siblings. The top left application of (Ret) occurs because an application of $(-\!\!<_R)$ to the bolded $p -\!\!< q$ is blocked, since its left premise would not be different from its conclusion. The key to finding the contradiction is the bolded $\mathbf{p} -\!\!< \mathbf{q}$ formula that is passed from the left-most leaf back to the right premise (1) of $(\to_R)$. The $(\bigwedge_R)$ in (1) is unary here, since the $\mathcal{P}$ variable contains only one set of formulae.

$$(\to_R)\;\dfrac{(\wedge_R)\;\dfrac{(-\!\!<_R)\;\dfrac{(Ret)\;\dfrac{}{\genfrac{}{}{0pt}{}{\mathcal{S}:=\{\{p,r,q\}\}}{\mathcal{P}:=\{\{\mathbf{p}-\!\!<\mathbf{q}\}\}}\Big|\Big|\; p, r, q \vdash \mathbf{p} -\!\!< \mathbf{q}} \qquad (Id)\;\dfrac{}{p, r \vdash p -\!\!< q, p}}{\genfrac{}{}{0pt}{}{\mathcal{S}:=\{\{p,r,q\}\}}{\mathcal{P}:=\{\{\mathbf{p}-\!\!<\mathbf{q}\}\}}\Big|\Big|\; p, r \vdash p -\!\!< q} \qquad (Id)\;\dfrac{}{p, r \vdash r}}{\genfrac{}{}{0pt}{}{\mathcal{S}:=\{\{p,r,q\}\}}{\mathcal{P}:=\{\{\mathbf{p}-\!\!<\mathbf{q}\}\}}\Big|\Big|\; p, r \vdash (p -\!\!< q) \wedge r} \qquad (1)}{p \vdash q, r \to ((p -\!\!< q) \wedge r)}$$

where (1) is:

$$(\textstyle\bigwedge_R)\;\dfrac{(-\!\!<_R)\;\dfrac{(Id)\;\dfrac{}{p, \mathbf{q} \vdash q, X, p -\!\!< q} \qquad (Id)\;\dfrac{}{p \vdash q, X, p -\!\!< q, \mathbf{p}}}{p \vdash q, X, \mathbf{p} -\!\!< \mathbf{q}}}{p \vdash q, X, \bigwedge\{\{\mathbf{p} -\!\!< \mathbf{q}\}\}}$$

We now show that proof search in **GBiInt** terminates because our soundness proof relies on the left premises of existentially branching rules to deliver variables to their right premises. The (Ret) rule is an *operational rule*. All other rules are *logical rules* and are categorised as follows: the *static rules* (Id), $(\bot_L)$, $(\top_R)$, $(\wedge_L)$, $(\vee_L)$, $(\wedge_R)$, $(\vee_R)$, $(\to_L)$, $(-\!\!<_R)$, $(\to^I_R)$ add formulae to the current world in the counter-model; the *transitional rules* $(\to_R)$, $(-\!\!<_L)$ create new worlds and add formulae to them; and the *special rules* $(\bigvee_L)$, $(\bigwedge_R)$ decompose variables returned from non-derivable leaves. This classification justifies our backward proof search strategy (Fig. 4).

```
Function Prove
Input: sequent $\gamma_0$
Output: Derivable ($true$ or $false$)

1. If $\rho \in \{(Id), (\bot_L), (\top_R)\}$ is applicable to $\gamma_0$ then return $true$
2. Else if any special or static rule $\rho$ is applicable to $\gamma_0$ then
   (a) Let $\gamma_1, \cdots, \gamma_n$ be the (universally branching) premises of $\rho$
   (b) Return $\bigwedge_{i=1}^{n} Prove(\gamma_i)$
3. Else for each transitional rule $\rho$ applicable to $\gamma_0$ do
   (a) Let $\gamma_1$ and $\gamma_2$ be the (existentially branching) premises of $\rho$
   (b) If $\bigvee_{i\in\{1,2\}} Prove(\gamma_i) = true$ then return $true$
4. Endif
5. Return $false$.
```

Fig. 4. Proof search strategy. Note that we have left out the variables for simplicity. $\bigwedge_{i=1}^{n} Prove(\gamma_i)$ is true iff $Prove(\gamma_i)$ is true for all premises γ_i for $1 \leq i \leq n$, and $\bigvee_{i\in\{1,2\}} Prove(\gamma_i)$ is true iff $Prove(\gamma_i)$ is true for some premise γ_i for $i \in \{1, 2\}$.

For a `BiInt` formula φ, the subformulae $sf(\varphi)$ are defined as usual with $\varphi \in sf(\varphi)$. For extended `BiInt` formulae $\bigvee \mathcal{S}$, $\bigwedge \mathcal{P}$, and a set Γ of extended `BiInt` formulae let:

$$sf(\bigvee \mathcal{S}) = \bigcup_{\Sigma \,\in\, \mathcal{S}} sf(\Sigma) \qquad sf(\bigwedge \mathcal{P}) = \bigcup_{\Pi \,\in\, \mathcal{P}} sf(\Pi) \qquad sf(\Gamma) = \bigcup_{\chi \,\in\, \Gamma} sf(\chi).$$

Note that the subformulae of $\bigvee \mathcal{S}$ and $\bigwedge \mathcal{P}$ do not include the conjunctions and disjunctions implicit in their `BiInt` equivalents.

Given a **GBiInt**-tree $\mathcal{T}$ and a branch $\mathcal{B}$ in $\mathcal{T}$, we say that $\mathcal{B}$ is **forward-only** if $\mathcal{B}$ contains only applications of static and special rules, $(\rightarrow_R)$ and the right premises of $(-\!\!<_L)$. Similarly, $\mathcal{B}$ is **backward-only** if $\mathcal{B}$ contains only applications of static and special rules, $(-\!\!<_L)$ and the right premises of $(\rightarrow_R)$. A branch is **single-directional** if it is either forward-only or backward-only. A branch contains **interleaved** left premises of transitional rules if it contains a sequence $\langle \cdots, \gamma_i, \cdots, \gamma_j, \cdots, \gamma_k, \cdots \rangle$ s.t. γ_i is the left premise of $(\rightarrow_R)$, γ_j is the left premise of $(-\!\!<_L)$, and γ_k is the left premise of $(\rightarrow_R)$.

Lemma 1. *Every forward/backward only branch of any* **GBiInt**-*tree is finite.*

Proof. We prove the lemma for forward-only branches, the one for backward-only branches is similar. Let $>_{len}$ be a lexicographic ordering of sequents: $(\Gamma_2 \vdash \Delta_2) >_{len} (\Gamma_1 \vdash \Delta_1)$ iff $|\Gamma_2| > |\Gamma_1|$, or ($|\Gamma_2| = |\Gamma_1|$ and $|\Delta_2| > |\Delta_1|$). Then from the blocking conditions of the rules, the length of a sequent according to $>_{len}$ increases on every forward-only branch: see [2] for details. Since **GBiInt** has the subformula property, eventually no more formulae can be added to a sequent on a forward-only branch, and the branch will terminate.

Lemma 2. *If a* **GBiInt**-*tree has an infinite branch, then the branch has an infinite number of interleaved left premises of transitional rules.*

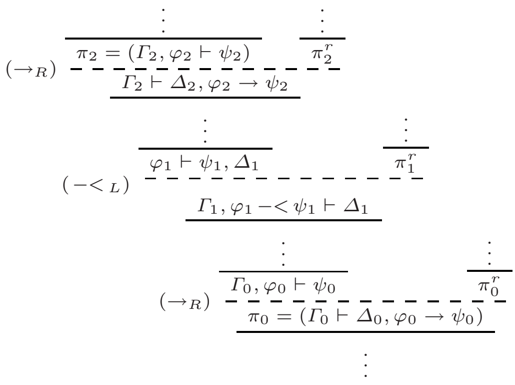

$\pi_2 = (\Gamma_2, \varphi_2 \vdash \psi_2)$
π_2^r
$(\to_R)$
$\Gamma_2 \vdash \Delta_2, \varphi_2 \to \psi_2$
$\varphi_1 \vdash \psi_1, \Delta_1$
π_1^r
$(-\!\!<_L)$
$\Gamma_1, \varphi_1 -\!\!< \psi_1 \vdash \Delta_1$
$\Gamma_0, \varphi_0 \vdash \psi_0$
π_0^r
$(\to_R)$
$\pi_0 = (\Gamma_0 \vdash \Delta_0, \varphi_0 \to \psi_0)$

Fig. 5. Interleaved left premises of transitional rules

Proof. By Lemma 1, single-directional branches must terminate. Thus, an infinite branch must involve an infinite number of interleaved left premises of transitional rules.

The degree of a BiInt formula φ is the number of $\to$ and $-\!\!<$ connectives in φ. The degree of a sequent $\Gamma \vdash \Delta$ is: $deg(\Gamma \vdash \Delta) = \sum_{\varphi \in sf(\Gamma \cup \Delta)} deg(\varphi)$. The following corollaries directly follow from the definition of the degree of a sequent.

Corollary 1. *By the subformula property of* **GBiInt***, the degree of a sequent can never increase in backward proof search.*

Corollary 2. *Given two sequents γ_1 and γ_2, if $sf(\gamma_2) \subsetneq sf(\gamma_1)$, then $deg(\gamma_2) < deg(\gamma_1)$. That is, removing some formula φ from a sequent during backward proof search decreases the sequent degree if φ is not a subformula of any other formula in the sequent.*

Theorem 1 (Termination). *Every* **GBiInt***-tree constructed according to the strategy of Fig. 4 is finite.*

Proof. Suppose for a contradiction that there exists an infinite **GBiInt**-tree $\mathcal{T}$. Since every rule has a finite number of premises, then by König's lemma $\mathcal{T}$ contains a branch $\mathcal{B}$ of infinite length. By Lemma 2, $\mathcal{B}$ contains an infinite number of interleaved left premises of transitional rules as shown in Fig. 5.

Let $\chi \in sf(\pi_0)$ be such that $deg(\chi) = max(\{deg(\varphi) \mid \varphi \in sf(\pi_0)\})$, that is, χ is one of the subformulae with the maximum degree. Thus χ is not a subformula of any formula with a larger degree. We show that $\chi \notin sf(\pi_2)$. There are two cases:

$\chi \notin sf(\Gamma_0)$**:** Then $\chi \in sf(\Delta_0)$ or $\chi = \varphi_0 \to \psi_0$. In both cases, $\chi \notin sf(\pi_2)$.

$\chi \in sf(\Gamma_0)$**:** Then it cannot be the case that $\chi \in sf(\varphi_1)$ or $\chi \in sf(\psi_1)$, since then $deg(\varphi_1 -\!\!< \psi_1) > deg(\chi)$, contradicting $deg(\chi) = max(\{deg(\varphi) \mid \varphi \in sf(\pi_0)\})$. Therefore, either:

- χ and all its occurrences in subformulae disappear from the sequent at the premise of $(-\!\!<_L)$, in which case $\chi \notin sf(\pi_2)$, or
- χ is moved to the RHS by applying $(\rightarrow_L)$ to some $\chi \rightarrow \tau$. But then $deg(\chi \rightarrow \tau) > deg(\chi)$, contradicting $deg(\chi) = max(\{deg(\varphi) \mid \varphi \in sf(\pi_0)\})$.

Thus we have $\chi \in sf(\pi_0)$ and $\chi \notin sf(\pi_2)$. Also, the subformula property of **GBiInt** gives $sf(\pi_2) \subseteq sf(\pi_0)$. Together with $\chi \in sf(\pi_0)$ and $\chi \notin sf(\pi_2)$, this means $sf(\pi_2) \subsetneq sf(\pi_0)$. Then by Corollary 2 we have $deg(\pi_2) < deg(\pi_0)$.

Note that the steps indicated by vertical ellipses ($\vdots$) in Fig. 5 are arbitrary, since by Corollary 1 no rule can increase the degree of a sequent. Since we have $deg(\pi_2) < deg(\pi_0)$, then every sequence of interleaved transitional rule applications must decrease the degree of the sequent. This can only happen a finite number of times, until no more transitional rules are applicable. Therefore our assumption was wrong, and no branch can be infinite. Hence every **GBiInt**-tree is finite.

4 Soundness

Instead of showing that each rule application preserves validity downwards, we show that each rule application preserves falsifiability upwards. Since variables introduce a two-way flow of information in the **GBiInt** trees, we separate the notion of soundness into two: *local soundness*, applicable to a single rule application, and *global soundness*, which considers the propagation of variables down the tree. Note that locality here refers to locality in the **GBiInt** trees, not locality in the underlying Kripke models.

Definition 2. *A logical rule in* **GBiInt** *is locally sound iff: if the conclusion is falsifiable, then some* ***universally*** *branching premise is falsifiable, or all* ***existentially*** *branching premises are falsifiable.*

Lemma 3. *Each static and special rule of* **GBiInt** *is locally sound.*

Proof. We assume the conclusion is falsifiable at w_0 in $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \vartheta \rangle$ and easily show that some premise is falsifiable at w_0 in $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \vartheta \rangle$: see [2] for details.

We now show global soundness, which relies on the notion of variable conditions because of the operational nature of **GBiInt**. Since our transitional rules use the variables returned from proof search of the left premise to instantiate the right premise, we need to show that the variables are instantiated and propagated soundly.

Lemma 4. *In any* **GBiInt** *tree $\mathcal{T}$, for every sequent $\gamma_0 \in \mathcal{T}$: if γ_0 is falsifiable, then some universally branching, or all existentially branching, premises are falsifiable, and the variable conditions hold at γ_0.*

Proof. By induction on the length $h(\gamma_0)$ of the longest branch rooted at γ_0.

Base case: $h(\gamma_0) = 0$. So γ_0 is an instance of (Id), $(\bot_L)$, $(\top_R)$, or (Ret).

- **(Id), $(\bot_L)$, $(\top_R)$:** The conclusion is never falsifiable, so there is nothing to show.
- **(Ret):** We show that the conclusion $\Gamma \vdash \Delta$ is always falsifiable, and that the variable conditions hold at $\Gamma \vdash \Delta$. We create a model with a single world w_0, and for every atom $p \in \Gamma$, let $\vartheta(w_0, p) = true$, and for every atom $q \in \Delta$, let $\vartheta(w_0, q) = false$. An atom cannot be both in Γ and Δ, since (Id) is not applicable to $\Gamma \vdash \Delta$.

  To show that $\Gamma \vdash \Delta$ is falsifiable at w_0, we need to show that $w_0 \vDash \Gamma$ and $w_0 \dashv \Delta$. For every atom in Γ and Δ, the valuation ensures this. For every composite formula φ, we do a simple induction on its length. Since (Ret) is only applied when no other rules are applicable, the required subformula ψ is already in Γ or Δ as appropriate, and ψ falls under the induction hypothesis. Thus we know that: (i) $w_0 \vDash \Gamma$ and (ii) $w_0 \dashv \Delta$. Then (i) and the persistence property of `BiInt` give us that $\forall w \in \mathcal{W} : w_0 \mathcal{R} w \Rightarrow w \vDash \Gamma$. Similarly, (ii) and the reverse persistence property of `BiInt` give us that $\forall w \in \mathcal{W}. w \mathcal{R} w_0 \Rightarrow w \dashv \Delta$. That is, the variable conditions hold at the conclusion of the (Ret) rule.

Induction step: We assume that the lemma holds for all γ_0 with $h(\gamma_0) \leq k$, and show that it holds for all γ_0 with $h(\gamma_0) \leq k+1$. Consider the rule application ρ such that γ_0 is the conclusion of ρ. By the assumption of the lemma, the conclusion γ_0 of ρ is falsifiable at some w_0 in some model $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \vartheta \rangle$. If ρ is a static or a special rule (universally branching), then the lemma easily follows from the induction hypothesis. Otherwise, ρ is a transitional rule (existentially branching). We show the case for $(\to_R)$, the case for $(-\!\!<_L)$ is symmetric:

$$
\begin{array}{c}
(\to_R)\ \dfrac{\begin{smallmatrix}\mathcal{S}_1\\ \mathcal{P}_1\end{smallmatrix}\Big\| \ \Gamma, \varphi \vdash \psi \qquad\qquad \begin{smallmatrix}\mathcal{S}_2\\ \mathcal{P}_2\end{smallmatrix}\Big\| \ \Gamma \vdash \Delta, \varphi \to \psi, \bigwedge \mathcal{P}_1}{\mathcal{S}/\mathcal{P} := \begin{cases} \mathcal{S}_1/\mathcal{P}_1 & \text{if } \mathcal{P}_1 = \epsilon \\ \mathcal{S}_2/\mathcal{P}_2 & \text{if right prem created} \\ \{\Gamma\}/\{\Delta, \varphi \to \psi\} & \text{otherwise} \end{cases} \Bigg\| \ \Gamma \vdash \Delta, \varphi \to \psi} \\
\text{right prem created only if } \mathcal{P}_1 \neq \epsilon \ \ \& \ \ \forall \Pi_i \in \mathcal{P}_1 . \Pi_i \not\subseteq \{\Delta, \varphi \to \psi\}
\end{array}
$$

Since the conclusion is falsifiable, there is a world w_0 such that (i) $w_0 \vDash \Gamma$ and, (ii) $w_0 \dashv \Delta, \varphi \to \psi$. From the `BiInt`-semantics of →, (ii) implies that there is a successor w_1 such that: (iii) $w_0 \mathcal{R} w_1$, (iv) $w_1 \vDash \varphi$ and (v) $w_1 \nvDash \psi$.

1. To show that the left premise γ_1 of $(\to_R)$ is falsifiable, we need to show that there exists a world w' s. t. γ_1 is falsifiable at w'. We let $w' = w_1$. Then (i), (iv) and (v) give us that γ_1 is falsifiable.

   Now, γ_1 has $height(\gamma_1) \leq k$, therefore the induction hypothesis applies to γ_1. By the induction hypothesis, since γ_1 is falsifiable at w_1, we have that the variable conditions hold at γ_1. In particular, the $\mathcal{P}$ condition holds, giving:

   $$\exists \Pi \in \mathcal{P}_1 . \forall w \in \mathcal{W}. w \mathcal{R} w_1 \Rightarrow w \dashv \Pi \tag{4.1}$$

   Now there are two cases: either the right premise γ_2 was created, or it was not (and there is nothing to show). If it was created, then we need to show that it is falsifiable by exhibiting a world w'' such that γ_2 is falsifiable at w''. We let $w'' = w_0$. Then, since $w_0 \mathcal{R} w_1$, we have $w_0 \dashv \Pi$ by (4.1). Since $\Pi \in \mathcal{P}_1$, then by the semantics of extended formulae, we have that $w_0 \dashv \bigwedge \mathcal{P}_1$. Together with (i) and (ii), this means that γ_2 is falsifiable at w_0. Moreover, the variable conditions hold at γ_2, since it also is falsifiable, and has $height(\gamma_2) \leq k$, so the induction hypothesis applies to it.

2. To show that the variable conditions hold at the conclusion γ_0, we do the case for $\mathcal{S}$; the case for $\mathcal{P}$ is dual. We need to show:

$$\exists \Sigma \in \mathcal{S}. \forall w \in \mathcal{W}. w_0 \mathcal{R} w \Rightarrow w \vDash \Sigma \tag{4.2}$$

Since we have shown that the variable conditions hold at the left premise, we know that in particular $\mathcal{P}_1 \neq \epsilon$. Then there are two cases: either the right premise was created, or it was not:

- If the right premise γ_2 was created, then its variable conditions hold, since γ_2 falls under the induction hypothesis. This gives us:

$$\exists \Sigma_2 \in \mathcal{S}_2. \forall w \in \mathcal{W}. w_0 \mathcal{R} w \Rightarrow w \vDash \Sigma_2.$$

Thus $\mathcal{S} := \mathcal{S}_2$ obeys (4.2).

- If the right premise was not created, then we need to show that the variable conditions hold at the conclusion for $S := \{\Gamma\}$. Now, we have $w_0 \vDash \Gamma$ by (i), and then the persistence property tells us that $\forall w \in \mathcal{W}. w_0 \mathcal{R} w \Rightarrow w \vDash \Gamma$. Thus $\mathcal{S} := \{\Gamma\}$ obeys (4.2).

Theorem 2 (Soundness). *If $\Gamma \vdash \Delta$ is derivable then $\Gamma \Vdash_{\texttt{BiInt}} \Delta$.*

Proof. We assume that $\Gamma \vdash \Delta$ is derivable and prove $\Gamma \vdash \Delta$ is not falsifiable. Then $\Gamma \Vdash_{\texttt{BiInt}} \Delta$ follows by definition. By induction on the height k of the derivation. **Base case:** A derivation of height 1 can only be an instance of $(\bot_L)$, $(\top_R)$ or (Id). In each case, γ is not falsifiable. **Inductive step:** We assume that if there is a derivation for γ of height $\leq k$, then γ is not falsifiable. Using Definition 1 and Lemma 4, it is easy to show by contradiction that if γ has a derivation of height $\leq k+1$, then γ is not falsifiable.

5 Completeness

We prove completeness via model graphs following [7]. We say that $\Gamma \vdash \Delta$ is **consistent** if $\bot \notin \Gamma$, $\top \notin \Delta$ and $\Gamma \cap \Delta = \epsilon$. We say that $\Gamma \vdash \Delta$ is **closed** w.r.t. a **GBiInt** rule ρ if either ρ is not applicable to $\Gamma \vdash \Delta$, or whenever $\Gamma \vdash \Delta$ matches the conclusion of an instance of ρ, then for some premise $\Gamma_1 \vdash \Delta_1$, we have $\Gamma_1 \subseteq \Gamma$ and $\Delta_1 \subseteq \Delta$. We say that $\Gamma \vdash \Delta$ is **saturated** if it is consistent and closed w.r.t. the static rules of **GBiInt**.

Corollary 3. *If $\frac{\mathcal{S}}{\mathcal{P}} \| \Gamma \vdash \Delta$ is not derivable, then $\Gamma \vdash \Delta$ is consistent for all $\mathcal{S}$ and $\mathcal{P}$.*

Remark 1. As usual, every sequent has a set of one or more "saturations" due to the branching of $(\wedge_R)$, $(\vee_L)$, etc., rules. The usual approach is to non-deterministically choose one of the non-derivable premises of each such rule. However, in the presence of the inverse relation, a branch that appears open may close once we return variables to a lower sequent. Therefore, we need to temporarily keep all the non-derivable premises, since we do not know which of the open branches will stay open when we return to a lower sequent.

Lemma 5. *For each finite non-derivable sequent $\Gamma \vdash \Delta$, there is an effective procedure to construct a finite set $\zeta = \{\alpha_1, \cdots, \alpha_n\}$ of finite saturated sequents, with $\Gamma \cup \Delta \subseteq LHS(\alpha_j) \cup RHS(\alpha_j) \subseteq sf(\Gamma) \cup sf(\Delta)$ for all $1 \leq j \leq n$.*

Proof. Let $\mathcal{T} = \Gamma \vdash \Delta$. Repeatedly apply static rules to the leaves of $\mathcal{T}$ to obtain new leaves. Keep the non-derivable leaves only; by Corollary 3 they are consistent. By Theorem 1, the saturation process will terminate; let $\zeta = \{\alpha_1, \cdots, \alpha_n\}$ be the final leaves of $\mathcal{T}$. By the subformula property, $LHS(\alpha_j) \cup RHS(\alpha_j) \subseteq sf(\Gamma) \cup sf(\Delta)$ for all $1 \leq j \leq n$.

Definition 3. *A model graph for a sequent $\Gamma \vdash \Delta$ is a finite* `BiInt` *frame $\langle \mathcal{W}, \mathcal{R} \rangle$ such that all $w \in \mathcal{W}$ are saturated sequents $\Gamma_w \vdash \Delta_w$ and:*

1. *$\Gamma \subseteq \Gamma_{w_0}$ and $\Delta \subseteq \Delta_{w_0}$ for some $w_0 \in \mathcal{W}$, where $w_0 = \Gamma_{w_0} \vdash \Delta_{w_0}$;*
2. *if $\varphi \rightarrow \psi \in \Delta_w$ then $\exists v \in \mathcal{W}$ with $w\mathcal{R}v$ and $\varphi \in \Gamma_v$ and $\psi \in \Delta_v$;*
3. *if $\varphi \prec \psi \in \Gamma_w$ then $\exists v \in \mathcal{W}$ with $v\mathcal{R}w$ and $\varphi \in \Gamma_v$ and $\psi \in \Delta_v$;*
4. *if $w\mathcal{R}v$ and $\varphi \rightarrow \psi \in \Gamma_w$ then $\psi \in \Gamma_v$ or $\varphi \in \Delta_v$;*
5. *if $v\mathcal{R}w$ and $\varphi \prec \psi \in \Delta_w$ then $\psi \in \Gamma_v$ or $\varphi \in \Delta_{w'}$;*
6. *if $w\mathcal{R}v$ and $\varphi \in \Gamma_w$ then $\varphi \in \Gamma_v$;*
7. *if $v\mathcal{R}w$ and $\varphi \in \Delta_w$ then $\varphi \in \Delta_v$.*

Lemma 6. *If there exists a model graph $\langle \mathcal{W}, \mathcal{R} \rangle$ for $\Gamma \vdash \Delta$, then there exists a* `BiInt` *model $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \vartheta \rangle$ such that for some $w_0 \in \mathcal{W}$, we have $w_0 \vDash \Gamma$ and $w_0 \dashv \Delta$. We call $\mathcal{M}$ the counter-model for $\Gamma \Vdash_{\texttt{BiInt}} \Delta$.*

Proof. Follows from Definition 3 by induction on the length of $\Gamma \vdash \Delta$.

We now show how to construct a model graph for $\Gamma \vdash \Delta$ from a consistent $\Gamma \vdash \Delta$. Recall from Remark 1 that we need to keep a number of independent versions of worlds because of the choices arising due to disjunctive non-determinism. We do this by storing one or more independent connected-components $\langle \mathcal{W}_1, \mathcal{R}_1 \rangle, \cdots, \langle \mathcal{W}_n, \mathcal{R}_n \rangle$ in the constructed model graph $\langle \mathcal{W}, \mathcal{R} \rangle$, and the indices (sorts) of worlds and relations tell us the connected-component of the graph to which they belong. We write $\langle \mathcal{W}_j, \mathcal{R}_j \rangle[j := i]$ to relabel the connected component $\langle \mathcal{W}_j, \mathcal{R}_j \rangle$ with sort j to a connected component $\langle \mathcal{W}_i, \mathcal{R}_i \rangle$ with sort i. Similarly, we also label each member of the variables $\mathcal{P}$ and $\mathcal{S}$, so we can later extract the member with sort i, corresponding to the component of $\langle \mathcal{W}, \mathcal{R} \rangle$ with sort i. We write $\mathcal{R}$-neighbour to mean $\mathcal{R}$-predecessor or $\mathcal{R}$-successor.

Our algorithm in Fig. 6 starts by saturating the root world to obtain one or more saturated "states". For each "state" α_i, it recursively creates all the

```
Procedure MGC
Input: sequent $\Gamma \vdash \Delta$
Output: model graph $\langle \mathcal{W}^f, \mathcal{R}^f \rangle$, variables $\mathcal{S}^f$ and $\mathcal{P}^f$

1. Let $\zeta = \{\alpha_1, \cdots, \alpha_n\}$ be the result of saturating $\Gamma \vdash \Delta$ using Lemma 5;
2. For each $\alpha_i \in \zeta$ do
  (a) Let $\langle \mathcal{W}_i, \mathcal{R}_i \rangle = \langle \{\alpha_i\}, \{(\alpha_i, \alpha_i)\} \rangle$; let $recompute := false$;
  (b) For each non-blocked $\varphi \to \psi \in \Delta_{\alpha_i}$ and while $recompute = false$ do
      i. Apply $(\to_R)$ to $\varphi \to \psi$ and obtain a left premise $\pi_1 = \Gamma_{\alpha_i}, \varphi \vdash \psi$;
     ii. Let $\langle \mathcal{W}, \mathcal{R} \rangle, \mathcal{S}, \mathcal{P} := MGC(\pi_1)$;
    iii. If $\exists \Pi_j \in \mathcal{P}. \Pi_j \subseteq \Delta_{\alpha_i}$ then
         A. Let $u_j \in \mathcal{W}_j$ be the root of the connected component $\mathcal{W}_j$ from $\mathcal{W}$;
         B. Let $G = \langle \mathcal{W}_j, \mathcal{R}_j \rangle [j := i]$; add $G$ to $\langle \mathcal{W}_i, \mathcal{R}_i \rangle$, and put $\alpha_i \mathcal{R}_i u_i$.
     iv. else
         A. Let $\langle \mathcal{W}_i, \mathcal{R}_i \rangle = \langle \epsilon, \epsilon \rangle$; let $recompute := true$;
         B. Invoke the right premise of $(\to_R)$ to obtain $\pi_2 = \Gamma_{\alpha_i} \vdash \Delta_{\alpha_i}, \bigwedge \mathcal{P}$;
         C. Apply $(\bigwedge_R)$ to $\pi_2$ to obtain $m \geq 1$ non-derivable premises $\gamma_1, \cdots, \gamma_m$;
         D. For each $\gamma_k$, $1 \leq k \leq m$, let $\langle \mathcal{W}_k, \mathcal{R}_k \rangle, \mathcal{S}_k, \mathcal{P}_k := MGC(\gamma_k)$;
         E. Let $\langle \mathcal{W}_i, \mathcal{R}_i \rangle := \langle \bigcup \mathcal{W}_k, \bigcup \mathcal{R}_k \rangle$, and $\mathcal{S}_i := \bigcup \mathcal{S}_{\gamma_k}$ and $\mathcal{P}_i := \bigcup \mathcal{P}_{\gamma_k}$;
  (c) For each non-blocked $\varphi -\!\!< \psi \in \Gamma_{\alpha_i}$ and while $recompute = false$ do
      i. Perform a symmetric procedure to Steps 2(b)i to 2(b)ivE.
  (d) If $recompute = false$ then let $\mathcal{S}_i := \{\Gamma_{\alpha_i}\}$ and $\mathcal{P}_i := \{\Delta_{\alpha_i}\}$.
3. Return $\langle \bigcup \mathcal{W}_i, \bigcup \mathcal{R}_i \rangle, \bigcup \mathcal{S}_i, \bigcup \mathcal{P}_i$
```

Fig. 6. Model Graph Construction Procedure

$\mathcal{R}$-neighbours and saturates them, and so on. If during the construction of any $\mathcal{R}$-neighbour, new information is returned from the higher sequents (Step 2(b)iv), then we delete the entire subtree (connected component of sort i) rooted at α_i, and recreate α_i using the new information (Step 2(b)ivB). This re-creates all the $\mathcal{R}$-neighbours of α_i. Otherwise, if none of the $\mathcal{R}$-neighbours of α_i return any new information, or there are no $\mathcal{R}$-neighbours for α_i, then Step 2d instantiates the variables and returns from the recursion. In the latter case, the "state" α_i already has all the required information it can possibly receive from any $\mathcal{R}$-neighbours, thus α_i is final. Note the duality: new information *from* a single $\mathcal{R}$-neighbour means that all of the members of a variable were new, while new information *at* a "state" α_i means that some $\mathcal{R}$-neighbour returned new information.

When we return from MGC, we form the union of the components of the model graph and the variables from the different "states", so that the caller of MGC can extract the appropriate component at Step 2(b)iiiA.

Remark 2. Note that while the counter-model construction procedure keeps the whole counter-model in memory, this procedure is only used to prove the completeness of **GBiInt**. Our procedure for checking the validity of `BiInt` formulae (Fig. 4) does not need the whole counter-model, and explores one branch at a time, as is usual for sequent/tableaux calculi.

Theorem 3 (Completeness). GBiInt *is complete: if $\Gamma \vdash \Delta$ is not derivable, then there exists a counter-model for $\Gamma \Vdash_{\mathtt{BiInt}} \Delta$.*

Proof. If $\Gamma \vdash \Delta$ is not derivable, then $\Gamma \vdash \Delta$ is consistent by Corollary 3. We construct a model graph for $\Gamma \vdash \Delta$ using the procedure of Fig. 6, and obtain $\langle \mathcal{W}^f, \mathcal{R}^f \rangle$. Let $\langle \mathcal{W}, \mathcal{R} \rangle$ be any connected component of $\langle \mathcal{W}^f, \mathcal{R}^f \rangle$. To show that $\langle \mathcal{W}, \mathcal{R} \rangle$ satisfies the properties of a model graph, we give the cases for properties 1, 2 and 4, the others are similar:

1. $\Gamma \subseteq \Gamma_{w_0}$ and $\Delta \subseteq \Delta_{w_0}$ for some $w_0 \in \mathcal{W}$: This holds because w_0 is one of the saturated sequents obtained from $\Gamma \vdash \Delta$. Moreover, if we delete the original w_0 at Step 2(b)ivA, a final version of w_0 is created at Step 2(b)iiiB which is never deleted.
2. if $\varphi \to \psi \in \Delta_w$ then $\exists v \in \mathcal{W}$ with $w\mathcal{R}v$ and $\varphi \in \Gamma_v$ and $\psi \in \Delta_v$: This holds because we have either created v using $(\to_R)$ at Step 2(b)iiiB, or had w fulfill the role of this successor by reflexivity if $(\to_R)$ was blocked.
4. if $w\mathcal{R}v$ and $\varphi \to \psi \in \Gamma_w$ then $\psi \in \Gamma_v$ or $\varphi \in \Delta_v$: In our construction, there are three ways of obtaining $w\mathcal{R}v$, so we need to show that for each case, the property holds. We first show that $\varphi \to \psi \in \Gamma_v$:
   1. v was created by applying $(\to_R)$ to w on some $\alpha \to \beta \in \Delta_w$. Then Γ_v also contains $\varphi \to \psi$.
   2. w was created by applying $(\mathrel{-\!\!<}_L)$ to some $\alpha \mathrel{-\!\!<} \beta \in \Gamma_v$. Then, when the final version of Γ_v was created, $\varphi \to \psi \in \Gamma_w$ was added to the $\mathcal{S}$ variable at Step 2d. There are two cases:
      - The right premise π_2 of $(\mathrel{-\!\!<}_L)$ was invoked at v. Then $\mathcal{S}$ was added to π_2 at v by the symmetric process to Step 2(b)ivB. Thus the updated Γ_v also contains $\varphi \to \psi$.
      - The right premise of $(\mathrel{-\!\!<}_L)$ was not invoked at v. This means that $\exists \Sigma_j \in \mathcal{S}.\Sigma_j \subseteq \Gamma_v$, and the j-th version of v's predecessor w is chosen at the symmetric process to Step 2(b)iiiA. But since Step 2d at w assigns $\Sigma_j := \Gamma_w$, then we have $\Gamma_w \subseteq \Gamma_v$ and thus $\varphi \to \psi \in \Gamma_v$.
   3. $v = w$, and $w\mathcal{R}w$ by reflexivity. Then $\Gamma_v = \Gamma_w$, so $\varphi \to \psi \in \Gamma_v$.

   In all cases, saturation for v will then ensure that $\psi \in \Gamma_v$ or $\varphi \in \Delta_v$.

We can obtain a counter-model for $\Gamma \Vdash_{\mathtt{BiInt}} \Delta$ from $\langle \mathcal{W}, \mathcal{R} \rangle$ via Lemma 6.

We use di-tree to mean a directed graph such that if the direction of the edges is ignored, it is a tree. The following corollary follows directly from our procedure since it never creates proper clusters: see [2].

Corollary 4. `BiInt` *is characterised by finite rooted reflexive and transitive di-trees.*

6 Conclusions and Future Work

Our cut-free calculus for `BiInt` enjoys terminating backward proof-search and is sound and complete w.r.t Kripke semantics. A simple Java implementation of **GBiInt** is available at `http://users.rsise.anu.edu.au/~linda`. The next step is to add a cut rule to **GBiInt**, and prove cut elimination syntactically. We are also extending our work to the modal logic `S5`, and the tense logic `Kt.S4`. Our approach of existential branching and inter-premise communication bears some similarities to hypersequents of Pottinger and Avron [1]. It would be interesting to investigate this correspondence further. From an automated deduction perspective, **GBiInt** is the first step towards an efficient decision procedure for `BiInt`. The next task is to analyse the computational complexity of **GBiInt** and investigate which of the traditional optimisations for tableaux systems are still applicable in the intuitionistic case.

We would like to thank the anonymous reviewers for their suggestions.

References

1. Avron, A.: The method of hypersequents in the proof theory of propositional non-classical logics. In: Proc. Logic Colloquium, Keele, UK, 1993, pp. 1–32, OUP (1996)
2. Buisman, L., Goré, R.: A cut-free sequent calculus for bi-intuitionistic logic: extended version (2007), `http://arxiv.org/abs/0704.1707`
3. Crolard, T.: Subtractive logic. Theor. Comp. Sci. 254(1–2), 151–185 (2001)
4. Czermak, J.: A remark on Gentzen's calculus of sequents. NDJFL, 18(3) (1977)
5. Dragalin, A.: Mathematical Intuitionism: Introduction to Proof Theory. Translations of Mathematical Monographs, vol. 68. Cambridge Univ. Press, Cambridge (1988)
6. Dyckhoff, R.: Contraction-free sequent calculi for intuitionistic logic. The Journal of Symbolic Logic 57(3), 795–807 (1992)
7. Goré, R.: Tableau methods for modal and temporal logics. In: D'Agostino, et al. (ed.) Handbook of Tableau Methods, pp. 297–396. Kluwer, Dordrecht (1999)
8. 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, Springer, Heidelberg (1996)
9. Horrocks, I., Sattler, U., Tobies, S.: A PSpace-algorithm for deciding $ALCNI_{R^+}$-satisfiability. LTCS-98-08, LuFG Theor. Comp. Sci, RWTH Aachen, (1998)
10. Howe, J.M.: Proof search issues in some non-classical logics. PhD thesis, University of St Andrews (1998)
11. Rauszer, C.: A formalization of the propositional calculus of H-B logic. Studia Logica 33, 23–34 (1974)
12. Rauszer, C.: An algebraic and Kripke-style approach to a certain extension of intuitionistic logic. Dissertationes Mathematicae, 168, Inst. of Math, Polish Academy of Sciences (1980)
13. Schwendimann, S.: A new one-pass tableau calculus for PLTL. In: de Swart, H. (ed.) TABLEAUX 1998. LNCS (LNAI), vol. 1397, Springer, Heidelberg (1998)
14. Śvejdar, V.: On sequent calculi for intuitionistic propositional logic. Commentationes Mathematicae Universitatis Carolinae 47(1), 159–173 (2006)
15. Szabo, M.E. (ed.): The Collected Papers of Gerhard Gentzen. Studies in Logic and the foundations of Mathematics. North-Holland, Amsterdam (1969)
16. Urbas, I.: Dual-intuitionistic logic. NDJFL 37(3), 440–451 (1996)
17. Uustalu, T.: Personal communication. via email (2004)
18. Uustalu, T.: Personal communication. via email (2006)
19. Uustalu, T., Pinto, L.: Days in logic '06 conference abstract. Accessed on 27th October 2006, `http://www.mat.uc.pt/~kahle/dl06/tarmo-uustalu.pdf`
20. Wolter, F.: On logics with coimplication. JPL 27(4), 353–387 (1998)

Tableaux with Dynamic Filtration for Layered Modal Logics[⋆]

Olivier Gasquet and Bilal Said

Université Paul Sabatier, IRIT - LILaC,
118 route de Narbonne, F-31062 Toulouse cedex 9, France

Abstract. In this paper we prove that the satisfiability problem for the class of what we call layered modal logics (LML) is in NEXPTIME, and hence, is decidable. Roughly, LML are logics characterized by semantical properties only stating the existence of possible worlds that are in some sense "further" than the other. Typically, they include various confluence-like properties, while they do not include density-like properties. Such properties are of interest for formalizing the interaction between dynamic and epistemic modalities for rational agents for example. That these logics are decidable may be not very surprising, but we show that they are all in NEXPTIME, some of them being known to be NEXPTIME-complete. For this, we give a sound and complete tableau calculus and prove that open tableaux are of exponential size. This cannot be done by using usual filtration which cannot cope with confluence. We introduce here a new technique we call *dynamic filtration* that allows to filtrate worlds one layer at a time keeping the total number of nodes within an exponential bound.

Keywords: Layered Modal Logic, Tableau, Dynamic Filtration, Complexity, Satisfaction problem.

1 Introduction

Some AI problems, like formalizing interaction of rational agents in the BDI (Belief-Desire-Intention) framework, or like complex ontologies require modal logics or description logics whose models possess complex properties which cannot be handled by tree-like structures (like it is the case for logics such as K,S4,S5,PDL,...). A typical example being the property associated with the "No forgetting" axiom ($\Box_1\Box_2 p \to \Box_2\Box_1 p$). Moreover experiment is needed and underlying logics can vary a lot. Hence, general results about decidability and complexity of such logics are of high interest. As well, theoretical tools that allow to investigate them are also of interest. At the moment, there are no general complexity results for such logics.

[⋆] This work has been partially supported by the project ARROWS of the French *Agence Nationale de la Recherche*.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 107–118, 2007.

Concerning theoretical tools, standard *filtration* ([10]) is one of them and is powerful enough to prove decidability (and even f.m.p.) of many standard logics (K,K4,K5,...), but for some complex logics, it often fails to establish decidability and even when it does, it does not give tight upper bounds on the complexity. Further refinements like *selective filtration* of [6] permits to state about various logics (e.g. [12] proves PSPACE-completeness of many extensions of K4), and more recently, *filtration via bisimulation* was defined and used in [13] to establish by a complex proof the f.m.p. of a wide class of multi-modal logics (mainly full and weak products) but without stating explicit upper bounds to the satisfiability problem. Tableaux are also a tool for this task, in [9] they first permitted to show that the complexity of S4 is in PSPACE (hardness being proved by a reduction to QBF). In the present work, tableaux are thus used more as a theoretical tool for investigating complexity of modal logics than as a way of designing tractable decision procedures.

Still, there are logics that are not in the scope of the methods mentionned above, a typical example is that of logic K plus confluence (corresponding to axioms $\Diamond\Box p \rightarrow \Box\Diamond p$) for which standard filtration fails. Tableaux may be easy to design for such logics, but they may be non-terminating and even if not, they may overestimate the complexity.

In this paper, we investigate logics we call *layered* and address their complexity by marrying tableaux with a stepwise filtration-like operation. This operation allows to keep the size of the tableau within an exponential of the length of the input formula. This bound is the best possible for the class of layered logics since one of them is known to be NEXPTIME-complete. Roughly speaking, layered logics are non-transitive confluent ones (where confluent is to be understood in a wide sense). They include properties like confluence (mono and multi modal), star-free PDL with converse and confluent programs, models containing bounded cycles, and we will see that with some adjustements they can cope with symmetry and converses, and with permutation. This work is on the continuation of the previous [3], [5], [4] and [7].

We believe that this technique may extend to other modal logics that are characterized by some class of frames which are directed acyclic grahs. This is subject of ongoing work.

In section 1, we give the necessary backgrounds, in section 2, we define layered modal logics, then in section 3 we design simple tableaux for these logics that we improve in section 4 to dynamically filtrated tableaux in order to prove the membership of the satisfaction problem for layered logics by means of tableaux calculi. We conclude with some discussion about the extension of the range of dynamic filtration.

2 Settings

Definition 1 (Language). *The language of a modal logic is defined by the following: let P be a set of propositional symbols, $\mathcal{I}$ a set of indexes, and as usual let*

$\bot$ *denotes falsity. The set* $FORM$ *of formulas (we will only consider* negative normal form *or NNF) is given by the BNF:*

$$\Phi ::= \bot|p|\neg p|(\Phi \wedge \Phi)|(\Phi \vee \Phi)|\Box_a\Phi|\Diamond_a\Phi (\textit{where } p \in P \textit{ and } a \in \mathcal{I})$$

and as usual, $(\_ \rightarrow \_)$ *abbreviates* $(\neg\_ \vee \_)$, $\Box_a^0\phi$ *is* ϕ *and* $\Box_a^{n+1}\phi$ *is* $\Box_a\Box_a^n\phi$. *Given a formula* ϕ, *we denote by* $|\phi|$ *the length of* ϕ.

Definition 2 (Modal degree). *The modal degree of a formula* ϕ *is denoted by* $d(\phi)$ *and is inductively defined as usual by:*

- $d(p) = d(\neg p) = d(\bot) = 0$ *(for any proposition* p*),*
- $d(\Box_a\phi) = d(\Diamond_a\phi) = d(\phi) + 1$,
- $d(\phi_1 \wedge \phi_2) = d(\phi_1 \vee \phi_2) = \max(d(\phi_1), d(\phi_2))$.
  The modal degree of a finite set S *of formulas is denoted by* $d(S)$ *and is equal to* $\max_{\phi\in S}(d(\phi))$.

Definition 3 (Subformulas). *As usual, given a formula* A, *the set of subformulas of* A *(in NNF) denoted by* $SF(A)$, *is recursively defined as:*

- $SF(p) = \{p\}$
- $SF(\bot) = \{\bot\}$,
- $SF(\neg p) = \{\neg p, p\}$,
- $SF(B * C) = \{B * C\} \cup SF(B) \cup SF(C)$ *(where* $* \in \{\wedge, \vee\}$*),*
- $SF(\Diamond_a B) = \{\Diamond_a B\} \cup SF(B)$ *and* $SF(\Box_a B) = \{\Box_a B\} \cup SF(B)$.

Definition 4 (Relations). *Given a relation* R *over a set* W, *we denote by* R^* *its reflexive and transitive closure, by* R^+ *its transitive closure, by* $-R$ *its converse (i.e.* $(x, y) \in -R$ *iff* $(y, x) \in R$*), and by* $(R\cup-R)$ *the symmetric closure of* R. *Finally, given a family (i.e. a set) of relations* $\mathcal{R} = \{R_a : a \in \mathcal{I}\}$, *we will also denote by* $\mathcal{R}$ *the relation consisting of the union of the relations of* $\mathcal{R}$, *i.e.* $\mathcal{R} = (\bigcup_{a\in\mathcal{I}} R_a)$.

Remark 1. We will also use the fact that a connected graph without isolated points (see below) may be represented by the set of its edges, i.e. by a conjunction of literals of the form $R_a(x, y)$.

Definition 5 (Semantics: frames, models and satisfaction)

- *(Multirelational Kripke)* frames *are graphs* $(W, \mathcal{R})$, *where* $\mathcal{R}$ *is a family of binary relations indexed by* $\mathcal{I}$, *and with a root* r*: any* $x \in W$ *is accessible from* r *via* $(\mathcal{R} \cup -\mathcal{R})^*$,
- *(Kripke)* models *are pairs* (F, m) *where* F *is a frame and* m *a meaning function (*$m: P \mapsto 2^W$*), such a model is said to be based on* F.
- Pointed models *are pairs* (M, x) *where* M *is a model* $(W, \mathcal{R})$ *and* $x \in W$.
- *That a formula* A *is* satisfied *by some pointed model (in symbols* $M, x \models A$*) is defined recursively as follows (we only give the clauses concerning modal connectives):*
  - $M, x \models \Diamond_a\psi$ *iff* $\exists y\colon R_a(x, y)\ \&\ M, y \models \psi$*;*
  - $M, x \models \Box_a\psi$ *iff* $\forall y\colon R_a(x, y) \Rightarrow M, y \models \psi$.

Definition 6 (Frame formula). *A* frame formula $\Phi(x_1, \ldots, x_n)$ *is a first-order formula (the x_i's are its free variables) which is a conjunction of literals $R_a(x, y)$ (with $a \in \mathcal{I}$), or equivalently a finite set of such literals.*

Definition 7 (Satisfiability Problem). *The satisfiability problem w.r.t. a class $\mathcal{C}$ of frames: given a formula A, does there exist some pointed model M, x based on some frame of $\mathcal{C}$ and such that $M, x \models A$? This problem is referred to as $\mathcal{C}$-satisfiability problem and the set of $\mathcal{C}$-satisfiable formulas will be denoted by $Sat(\mathcal{C})$.*

Definition 8 (Vector notation). *For the sake of brevity, we introduce here a vector notation which is as follows: (given the variables $x_1, \ldots, x_n$, the variable x, the functions $H_1, \ldots, H_m$ and the function H)*

- *The sequence $(x_1, \ldots, x_n)$ will be abbreviated by $\overrightarrow{x}$;*
- *$H(x_1, \ldots, x_n)$ will be abbreviated by $H(\overrightarrow{x})$;*
- *the sequence $(H(x_1), \ldots, H(x_n))$ will be abbreviated by $H.\overrightarrow{x}$; (note the dot)*
- *the sequence $(H_1(x), \ldots, H_m(x))$ will be abbreviated by $\overrightarrow{H}.x$;*
- *and the sequence $(H_1(\overrightarrow{x}), \ldots, H_m(\overrightarrow{x}))$ will be abbreviated by $\overrightarrow{H}.\overrightarrow{x}$.*

Definition 9 (Subframe/subgraph). *Given a frame $F = (W, \mathcal{R})$, a first-order formula $\Phi(\overrightarrow{x})$ whose free variables are $x_1, \ldots, x_n$ and given an assignment $\sigma : \{x_1, \ldots, x_n\} \mapsto W$, we consider that $\sigma(F)$ denotes the subframe (w, r) of F where $w = \{\sigma(x_1), \ldots, \sigma(x_n)\}$ and $r_a = (R_a)_{|w}$ (i.e. the restriction of R_a to w). We say that the subframe $\sigma(F)$ satisfies Φ (in symbols $F \models \Phi(\sigma.\overrightarrow{x})$) iff $\Phi(\sigma(x_1), \ldots, \sigma(x_n))$ is true in F.*

Definition 10 (Depth). *Given a frame $F = (W, \mathcal{R})$ with root r represented by a set S of literals $R_a(x, y)$ or by a frame formula Φ, given $x \in W$, we define the* depth *in F (or in S) denoted by $\delta_F(x)$ (or $\delta_S(x)$ or $\delta_\Phi(x)$) as the length of the shortest path from r to x. Inductively:*

- $\delta_F(r) = 0$;
- $\delta_F(x) = \min_{R_a(y,x) \in \mathcal{R}}(\delta_F(y) + 1)$

3 Layered Modal Logics

We come to the class of frames we investigate:

Definition 11. *An LF-*layered *frame $F = (W, \mathcal{R})$ is a finite frame (of root r) which verifies a finite set LF of* layer formulas *of the form:*

$$\forall \overrightarrow{x} : \exists \overrightarrow{y} : \phi(\overrightarrow{x}) \rightarrow \psi(\overrightarrow{x}, \overrightarrow{y})$$

where ϕ and ψ are frame formulas and with the constraints given below. But first notice that such a formula may be seen as a rule which rewrites a graph by adding nodes and edges to it, ϕ describing the left-hand side of the rule (the graph to be rewritten) while $\phi \wedge \psi$ describes the right-hand side (the result of the rewriting). Constraints on ϕ and ψ are the following:

(i) $\phi(\overrightarrow{x})$ *is a conjunction (that we will identify with a set) of literals* $R_a(x_i, x_j)$ *(where* $x_i, x_j \in \overrightarrow{x}$ *and* $a \in \mathcal{I}$*);*

(ii) $\psi(\overrightarrow{x}, \overrightarrow{y})$ *is a conjunction of literals* $R_a(x_i, y_j)$ *or* $R_a(y_j, y_k)$ *(where* $x_i \in \overrightarrow{x}$*,* $y_j, y_k \in \overrightarrow{y}$*,* $j < k$ *and* $a \in \mathcal{I}$*);*

(iii) $\forall y_j \in \overrightarrow{y} : \exists x_i \in \overrightarrow{x} : \exists a \in \mathcal{I}$: $(R_a(x_i, y_j))$ *is a conjunct of* ψ*; we exclude properties stating the existence of isolated nodes;*

(iv) $\delta_{\phi\wedge\psi}(y_k) > \delta_{\phi\wedge\psi}(x_j)$ *for all* $y_k \in \overrightarrow{y}$ *and all* $x_j \in \overrightarrow{x}$*: the depth of existential nodes will always be stricly greater than that of their parents nodes, hence, their modal degree will be stricly less.*

Examples of such layered formulas are:

1. Seriality $(\forall x \colon \exists y R(x, y))$;
2. Bimodal confluence $(\forall x, y, z \colon \exists u \colon (R_1(x, y)\& R_2(x, z)) \rightarrow (R_2(y, u)\& R_1(z, u)))$.

Definition 12. *A SL-formula (skolemized layer formula)* $skol(\chi)$ *is the result of the skolemization of a layered formula* $\chi = \forall \overrightarrow{x} : \exists \overrightarrow{y} : \phi(\overrightarrow{x}) \rightarrow \psi(\overrightarrow{x}, \overrightarrow{y})$*:*

$$skol(\chi) = \forall \overrightarrow{x} : \phi(\overrightarrow{x}) \rightarrow \psi(\overrightarrow{x}, \overrightarrow{H}.\overrightarrow{x})$$

with $\overrightarrow{H}.\overrightarrow{x} = (H_1(\overrightarrow{x}), \ldots, H_k(\overrightarrow{x}))$ *(and* $k = \mathrm{Card}(\overrightarrow{y})$*) Since* ψ *is a conjunction,* $skol(\chi)$ *is equivalent to the conjunction of formulas of the form*

(i) either $\forall \overrightarrow{x} : \phi(\overrightarrow{x}) \rightarrow R_a(x_i, H_j(\overrightarrow{x}))$

(ii) or $\forall \overrightarrow{x} : \phi(\overrightarrow{x}) \rightarrow R_a(H_i(\overrightarrow{x}), H_j(\overrightarrow{x}))$*.*

We will henceforth consider SL-formulas as being of one of the forms (i) and (ii).

By extension, SLF is the set of type (i) and type (ii) formulas corresponding to a set LF of layered formulas.

We will see further that $\Diamond$ formulas also are treated by the introduction of Skolem functions.

The SLF's for

- Seriality it is $\forall x R(x, H(x))$
- Confluence they are $\forall x, y, z \colon ((R_1(x, y) \;\&\; R_2(x, z) \rightarrow R_2(y, H(x, y, z)))$ and $\forall x, y, z \colon ((R_1(x, y) \;\&\; R_2(x, z)) \rightarrow R_1(z, H(x, y, z)))$;

Remark: It is well-known that $A \in Sat(LF)$ iff $A \in Sat(skol(LF))$, since skolemization preserves satisfiability.

Definition 13 (Layered logics). Layered *logics are those characterized by a class of SLF-layered frames i.e. finite rdag veryfing a set SLF of SL-formulas of type (i) and (ii).*

That a given logic is or is not a layered one is a natural question (note that layered logics are semantically defined). For example, we know that modal logic K+confluence is characterized by the class of confluent frames, and since the first-order formula expressing confluence is a layer formula then K+confluence is a layered logic. But in general, given a set of properties, checking if they are equivalent with some set of layer formulas is likely to be undecidable although we did not formally prove this.

The main result of the next sections is that the SLF-satisfiability problem for any SLF-layered logic is in NEXPTIME and hence is decidable. More precisely, it is decidable by a non-deterministic Turing machine in time $O(2^{c.|\phi|})$ where c is a constant.

4 Simple Tableaux for Layered Logics

Tableaux calculi may standardly be seen either as a sequent-like construction (see e.g. [8] and [11]) or as a stepwise "quasi-model" construction as done in e.g. [1]. We adopt this second view in the following. This step-by-step construction may be seen as rewriting a graph, starting from some initial node containing the input formula, and using some appropriate set of rewriting rules (or simply rules) which *only add* elements (formulas, nodes, edges) to the current structure.

Definition 14 (Partial tableau). *The structure called a* partial tableau *is a triple $T = (N, \mathcal{R}, \Phi)$ where: N is a finite set of nodes, $\mathcal{R}$ is a family of binary relations over N and indexed by $\mathcal{I}$ (hence $(N, \mathcal{R})$ is a finite frame) and Φ is a function that maps each element of N to some set of formulas ($\Phi\colon N \to 2^{FORM}$). Given two such functions Φ and Ψ, we define $\Phi \cup \Psi$ by: $\Phi \cup \Psi(x) =$*

- $\Phi(x) \cup \Psi(x)$ *if* $x \in \mathrm{dom}(\Phi) \cap \mathrm{dom}(\Psi)$;
- $\Phi(x)$ *if* $x \in \mathrm{dom}(\Phi)$ *and* $x \notin \mathrm{dom}(\Psi)$;
- $\Psi(x)$ *if* $x \in \mathrm{dom}(\Psi)$ *and* $x \notin \mathrm{dom}(\Phi)$.

A tableau is a partial tableau on which no rule applies, more precisely a tableau is the least fixed point of a sequence of partial ones.

Definition 15 (Skolemizing $\Diamond$). *Let (N, R, Φ) be a partial tableau for the input formula A, and let $SF(A)$ denote the set of subformulas of A (def. 3). For each triple $(a, x, B) \in \mathcal{I} \times N \times SF(A)$, we associate a Skolem term $\Diamond_a(B, x)$ (intuitively this term will denote one world accessible from x and making B true, thus making $\Diamond_a B$ true at x).*

A rule may be seen as a function ρ applied to a partial tableau T_i and computing what new elements (denoted by $\nu_\rho(T_i)$) are to be added to T_i in order to obtain T_{i+1} (in the case of rule $\vee$ this function non-deterministically chooses one of the possible results) and this function gives a triple $\nu_\rho(T_i) = (n, r, f)$ (respectively the sets of new nodes, new a-edges (for each $a \in \mathcal{I}$) and new pairs node-formula).

Definition 16 (Rule application). *Let $T_i = (N_i, \mathcal{R}_i, \Phi_i)$ be a partial tableau, and let ρ be a rule, to denote that T_{i+1} is obtained from T_i by applying rule ρ, we write:*

$$T_{i+1} = \rho(T_i) = T_i \cup \nu_\rho(T_i)$$

Definition 17 (Set of rules). *For each possible rule ρ, we indicate the result of $\nu_\rho(T_i)$ as a triple (n_i, r_i, f_i). Variables u and v are implicitely universally quantified over N_i. Moreover, we indistinctly use $B \in \Phi(u)$ or $(u, B) \in \Phi$.*

- $\nu_\bot(T_i) = \Big(\emptyset, \emptyset, \{(u, \bot)\}\Big)$ *for all* $B, \neg B \in \Phi_i(u)$
- $\nu_\wedge(T_i) = \Big(\emptyset, \emptyset, \{(u, B), (u, C)\}\Big)$ *for all* $(B \wedge C) \in \Phi_i(u)$
- $\nu_\vee(T_i) = \Big(\emptyset, \emptyset, \{(u, D_{u,B\vee C})\}\Big)$ *for all* $(B \vee C) \in \Phi_i(u)$
  if $B, C \notin \Phi_i(u)$ *then* $D_{u,B\vee C}$ *is choosen non-deterministically among* B *and* C,
  else $D_{u,B\vee C}$ *is any of* B *and* C *which is already in* $\Phi_i(u)$
- $\nu_{\Box_a}(T_i) = \Big(\emptyset, \emptyset, \{(v, B)\}\Big)$ *for each* $a \in \mathcal{I}$, *for all* u, v *such that* $R_a(u, v)$ *and* $\Box_a B \in \Phi_i(u)$
- $\nu_{\Diamond_a}(T_i) = \Big(\{\Diamond_a(B, u)\}, \{(u, \Diamond_a(B, u))\}, \{(\Diamond_a(B, u), B)\}\Big)$ *for each* $a \in \mathcal{I}$ *and for all* $\Diamond_a B \in \Phi_i(u)$; *NB:* $\delta(\Diamond_a(B, u)) > \delta(u)$.
- *If the formula* $\psi = \forall \overrightarrow{x} : \Phi(\overrightarrow{x}) \to R_a(x_k, H_l(\overrightarrow{x}))$ *is in* SLF,
  and if for some assignment σ *of* $\overrightarrow{x}$ *over* N_i *we have* $T_i \models \Phi(\sigma.\overrightarrow{x})$,
  then
    if $\exists x_j \in \overrightarrow{x} : d(\Phi_i(\sigma(x_j))) > 0$ [1]
    then $\nu_\psi(T_i) = \Big(\{H_l(\sigma.\overrightarrow{x})\}, \{(\sigma(x_k), H_l(\sigma.\overrightarrow{x}))\}, \emptyset\Big)$
    NB: $\delta(H_l(\sigma.\overrightarrow{x})) > \delta(\sigma(x_j))$;
    else $= \Big(\emptyset, \{(\sigma(x_k), \sigma(x_j))\}, \emptyset\Big)$ *for each* $x_k, x_j \in \overrightarrow{x}$;
- *If the formula* $\psi = \forall \overrightarrow{x} : \Phi(\overrightarrow{x}) \to R_a(H_k(\overrightarrow{x}), H_l(\overrightarrow{x}))$ *is in* SLF,
  and if for some assignment σ *of* $\overrightarrow{x}$ *over* N_i *we have* $T_i \models \Phi(\sigma.\overrightarrow{x})$,
  then
    if $\exists x_j \in \overrightarrow{x} : d(\Phi_i(\sigma(x_j))) > 0$
    then $\nu_\psi(T_i) = \Big(\{H_k(\sigma.\overrightarrow{x}), H_l(\sigma.\overrightarrow{x})\}, \{(H_l(\sigma.\overrightarrow{x}), H_l(\sigma.\overrightarrow{x}))\}, \emptyset\Big)$
    NB: $\delta(H_l(\sigma.\overrightarrow{x})) > \delta(H_k(\sigma.\overrightarrow{x_j})) > \delta(x_j)$;
    else $= \Big(\emptyset, \{(\sigma(x_k), \sigma(x_j))\}, \emptyset\Big)$ *for each* $x_k, x_j \in \overrightarrow{x}$.

Remark 2.

- If A is the input formula, $\forall x \in N_i : \Phi_i(x) \subseteq SF(A)$.
- Since the set SLF is finite, so are each n_i.
- In the last two "else" part, no new node is added (n_i is empty).

Definition 18 (Meta-rules). *A direct examination of the rules above shows that all rules* ρ *except the diamond rule and those of the form* ν_ψ *for* $\psi \in SLF$ *are terminating on partial (and hence finite) tableaux, i.e. there exists an integer* n *such that* $\rho^n(T_i) = \rho^{n+1}(T_i)$ *(where* $\rho^{n+1}(T_i) = \rho(\rho^n(T_i))$*), we denote by* ρ^* *the iteration of* ρ *up to the least fixed point in these cases.*
Let us define the following meta-rules called Sat$\Box$, $\Diamond$ *and* C:

- $\nu_{\mathrm{Sat}\Box} = (\nu_\bot \cup \nu_\wedge \cup \nu_\vee \cup (\bigcup_{a\in\mathcal{I}} \nu_{\Box_a}))^*$ *(classical saturation and* $\Box$ *propagation: terminates since frames and formulas are of finite size)*,
- $\nu_\Diamond = (\bigcup_{a\in\mathcal{I}} \nu_{\Diamond_a})$ *(without star!)*,
- $\nu_C = (\bigcup_{\psi\in SLF} \nu_\psi)$.

[1] This condition stops the computation when nodes only contains non modal formulas.

Definition 19 (Simple tableaux). *A simple tableau T for an input formula A is a least fixed point (we will prove later that this fixed point is finite) of a sequence $T_0 = (N_0, \mathcal{R}_0, \Phi_0), T_1, \ldots$ where*[2]*:*

- *$N_0 = \{r\}$ (the root), $\mathcal{R}_0 = \emptyset$ and $\Phi_0(r) = \{A\}$;*
- *$T_{i+1} = \mathrm{Sat}\Box(C(\Diamond(T_i)))$, where C, $\mathrm{Sat}\Box$ and $\Diamond$ are the meta-rules given above.*

Definition 20 (Tableau closure). *A tableau is closed if some node in it contains $\bot$; it is open otherwise.*

Definition 21 (First occurrence). *Let $x \in N$, let $\mathbf{fst}(x) = \min_j(x \in N_j))$, this is the index of introduction of x in N. Hence $x \in N_i$ iff $i \geq \mathbf{fst}(x)$.*

Proposition 1. *Let A be an input formula ($d(A)$ being its modal degree), and let $T = (N, \mathcal{R})$ be a (possibly partial) simple tableau from a sequence $T_0, \ldots, T_n, \ldots$ with root r:*

1. *For all $j > i \geq \mathbf{fst}(y)$ we have $\delta_{T_i}(y) = \delta_{T_j}(y)$: once set, the depth of a node do not change at further iteration, we will then denote it by $\delta(y)$;*
2. *for all $j > i \geq \mathbf{fst}(y)$ we have $d_{T_i}(y) = d_{T_j}(y)$: idem but for the modal degree of nodes;*
3. *if $\delta(x) = \min_{x_k \in \overrightarrow{x}}(\delta(x_k))$ and $y = H(\overrightarrow{x})$ then $d(y) \leq d(x) - 1$;*
4. *for all y we have $\mathbf{fst}(y) \leq \delta(y)$;*
5. *for all y we have $0 \leq d(y)+\delta(y) \leq d(A)$ and as a consequence $\mathbf{fst}(y) \leq d(A)$: the algorithm stops after at most $d(A)$ iterations.*

Proof.

1. Since shortest path cannot shrink due to constraints on rules.
2. Since classical saturation and propagation of boxed formulas are performed at each iteration, and since nodes introduced later are of strictly greater depth.
3. Since rules that add formulas to new nodes ($\Diamond$ and $\Box$ rules) stricly decrease the modal degree of formulas; thus along a path (r, y) of length p, node y will only receive formulas of degree at most $d(A) - p$. Hence, formulas of highest degree in y will come from a shortest path, i.e. from x.
4. By induction on $\mathbf{fst}(y)$: If $\mathbf{fst}(y) = 0$ then $y = r$ and $\delta(r) = 0$; if $\mathbf{fst}(y) = p+1$ then there exist $x_1, \ldots, x_n \in N_p$ and $x_k \notin N_{p-1}$ (otherwise y would be in N_p) with $y = H(\overrightarrow{x})$. Then since $\delta(y) > \delta(x_k)$ and $\delta(x_k) \geq \mathbf{fst}(x_k) = p$ (by IH) hence $\delta(y) \geq p + 1$.
5. Again by induction on $\mathbf{fst}(y)$. If $\mathbf{fst}(r) = 0$ then $y = r$, $d(r) = d(A)$ and $\delta(r) = 0$: done. Suppose $\mathbf{fst}(x) = p+1$ then for some $x_1, \ldots, x_n$ we must have $x = H(\overrightarrow{x})$. Let $\delta(x) = \min_{x_k \in \overrightarrow{x}}(\delta(x_k))$, rules ensure that $\delta(y) = \delta(x) + 1$ and since $d(y) \leq d(x) - 1$ we have $d(y) + \delta(y) \leq d(x) + \delta(x) \leq d(A)$ (by IH).

[2] Due to rule $\vee$, the are several such sequences.

Lemma 1 (Completeness and soundness of simple tableaux). *Let A be a formula and S a set of SLF-formulas, then A is SLF-satisfiable if and only if there exists an open tableau $T = (\mathrm{Sat}\Box(C(\Diamond)))^*(T_0)$ where C is the set of rules corresponding to the set S.*

Proof. Completeness is immediate by comparison with naive tableaux (those with purely non-deterministic but fair strategy where all rules are applied only *non-deterministically* but *eventually at some iteration*) which are trivially complete since they reduce to model construction. Now, it can be checked that the strategy $\mathrm{Sat}\Box(C(\Diamond))^*$ is fair hence if some simple tableau is open then so is some naive one and completeness follows from that of naive tableaux.

Soundness is easily proved by induction on iterations and with the help of proposition 1. It remains to prove that frame of an open simple tableau is an SLF-frame: we have to prove that $(W, \mathcal{R}) \models SLF$. Let $\psi = (\forall(\phi(\overrightarrow{x}) \rightarrow R_a(x_j, H_k(\overrightarrow{x})))$ be in SLF, and suppose that for some assignment σ over W, we have $(W, \mathcal{R}) \models \phi(\sigma.\overrightarrow{x})$ and let $m = \max_{x_i \in \overrightarrow{x}}(\mathbf{fst}(\sigma(x_i))) \subseteq N_q)$ (m is the first iteration at which the all elements of $\sigma.\overrightarrow{x}$ have been introduced). There are two cases:

(a) If $\exists x_p\colon \sigma(x_p) \in N_m$ & $d(\Phi_m(\sigma(x_p))) > 0$ then $(\sigma(x_j), H_k(\sigma.\overrightarrow{x})) \in (R_a)_{m+1}$ by then-part of rule ν_ψ, hence $(\sigma(x_j), H_k(\sigma.\overrightarrow{x})) \in R_a$;
(b) If $\forall x_p\colon \sigma(x_p) \in N_m \Rightarrow d(\Phi_m(\sigma(x_p))) = 0$ then we are done since $(\sigma(x_j), \sigma(x_j)) \in R_a$ by else-part of the same rule.

A similar reasoning holds in the case of SL-formulas of type ii since:
in case (a) $(H_j(\sigma.\overrightarrow{x}), H_k(\sigma.\overrightarrow{x})) \in R_a$ by then-part,
and in case (b) $(\sigma(x_j), \sigma(x_k)) \in R_a$ by else-part.

Lemma 2. *Layered modal logics are decidable by proposition 1 and by completeness and soundness.*

Proof. Since only finitely many nodes are added at each iteration and since there are at most $d(A)$ iterations.

On another hand, at least one layered modal logic (e.g. K+confluence, see [7]) is known to have an NEXPTIME-hard satisfiability problem. Hence the best we can do for this class of logics is to prove their membership in NEXPTIME. In order to do so, we develop and apply, in the sequel, tableaux with dynamic filtration.

5 Dynamically Filtrated Tableaux

Starting from a partial tableau $T_i = (N_i, \mathcal{R}_i, \Phi_i)$, rules of simple tableaux add new nodes (set n_i) and edges (set r_i) at iteration $i + 1$ in order to fulfill semantics of $\Diamond$ connectives and existential properties stated by SL-formulas. But for proving our complexity result, we need to make sure that each node added is unique otherwise we would have multiple copies of the same node (i.e. of its associated set of formulas). Thus at iteration $i + 1$ we identify those new nodes when they are equivalent w.r.t. the input formula. This justify the name of dynamic filtration.

Definition 22 (Dynamic filtration). *Let T_i be a partial tableau $T_i = (N_i, \mathcal{R}_i, \Phi_i)$ (NB: in what follows we will most of the time omit the index i and denote T_i by $T, \dots$) and let $T' = (N \cup n, \mathcal{R} \cup r, \Phi \cup f) = \mathrm{Sat}\Box(C(\Diamond(T)))$.*

Let us define the following equivalence relation $\equiv^i$ (we will also omit the exponent i henceforth) over $N \cup n$:
$x \equiv y$ iff and $f(x) = f(y)$ for $x, y \in n$ and $x = y$ for $x \in N$ (this is just to make $\equiv$ total over $N \cup n$),
$\equiv (x)$ denotes the equivalent class of x;
$(n)_\equiv = \{\equiv (x) \colon x \in n\}$ is the set of all classes $\equiv (x)$.

Then we set the dynamic filtration *of T' to be $DF(T') = (N', \mathcal{R}', \Phi')$ with:*

- *$N' = N \cup (n)_\equiv$;*
- *$\mathcal{R}' = \mathcal{R} \cup (\equiv \circ\, r \circ equiv)$ [3], hence an edge $(\equiv (x), \equiv (y))$ is added to $\mathcal{R}$ either if $x \in N$ (since then $\equiv (x) = x$) and $(x, y) \in r$, or if $x, y \in n$ and $(x, y) \in r$;*
- *$\Phi' = \Phi \cup (f)_{|(n)_\equiv}$.*

Definition 23 (Filtrated tableaux). *A filtrated tableau T for an input formula A is a least fixed point (we will prove later that this fixed point is finite) of a sequence $T_0 = (N_0, \mathcal{R}_0, \Phi_0), T_1, \dots$ where[4]:*

- *$N_0 = \{r\}$ (the root), $\mathcal{R}_0 = \emptyset$ and $\Phi_0(r) = \{A\}$;*
- *$T_{i+1} = DF(\mathrm{Sat}\Box(C(\Diamond(T))))$, where C, $\mathrm{Sat}\Box$ and $\Diamond$ are the meta-rules given above and DF is the dynamic filtration operation.*

Proposition 2. *Proposition 1 holds as it is for filtrated tableaux as well, as it can easily be checked.*

Proposition 3. *As a direct consequence of the definitions above, at each iteration, only an exponential number of nodes is added:*
$\mathrm{Card}(\{x \colon x \in N_i \;\&\; \mathbf{fst}(x) = k \textit{ for some } k \le i\}) \le \mathrm{Card}(\equiv^i) \le 2^{\mathrm{Card}(SF(A))} = 2^{c.|A|}$ *for some constant c.*

Lemma 3. *Let $T = (N, \mathcal{R}, \Phi)$ be an open tableau for A then T contains at most exponentially many nodes each of size bounded by $|A|$.*

Proof. From proposition 1, $\forall x \in N \colon 0 \le \mathbf{fst}(x) \le d(A) \le |A|$ and by prop. 3 there are at most $2^{c.|A|}$ nodes for each iteration, hence T contains at most $|A|.2^{c.|A|}$ nodes which is majorated by $2^{c'.|A|}$ for some constant c'. Note that nodes are of maximal size $|A|$ since $\Phi(x) \subseteq SF(A)$.

Lemma 4 (Soundness and completeness). *Filtrated tableaux are sound and complete for layered modal logics.*

Proof. Let $U = (N^U, \mathcal{R}^U, \Phi^U)$ be a simple tableau fixed point of a sequence $U_0, \dots, U_n$ and let $T = (N^T, \mathcal{R}^T, \Phi^T)$ be the filtrated tableau fixed point of the sequence $DF(U_0), \dots, DF(U_n)$. Let $M_U = (N^U, \mathcal{R}^U, m^U)$ be the Kripke model

[3] Where $\circ$ denotes composition: $R \circ S(x, y) \Leftrightarrow \exists z : R(x, z) \;\&\; S(z, y)$.

[4] Due to rule $\vee$, the are several such sequences.

defined by $m^U(p) = \{x \in N^U : p \in \Phi^U(x)\}$ and M_T defined similarly. If we set the function π as $\pi(x) = \equiv_{\mathbf{fst}} (x)$ from M^U onto M^T, it is straightforward to check that π is a p-morphism (i.e. a pseudo-epimorphism) from M_U to M_T: $x \in N^U$ and $\pi(x) \in N^T$ satisfy the same propositions, and $(x, y) \in \mathcal{R}^U$ iff $(\pi(x), \pi(y)) \in \mathcal{R}^T$, thus x and $\pi(x)$ satisfy the same formulas. Hence filtrated tableaux are sound and complete for layered logics since so are simple tableaux.

As a direct consequence of lemmas 4 and 3, we have:

Theorem 1. *SLF-satisfiability is decidable in non-deterministic exponential time.*

Proof. The following non-deterministic algorithm does the job: guess a tableau for A and check whether it is open (this can be done in linear time in the size of the tableau, hence in time bounded by $O(2^{c.|\phi|})$.

6 Discussion and Conclusion

Some properties cannot directly be considered as layered ones, we discuss some of them and show how they can still be handled.

- The case of symmetry. In order to treat such a property, the rule $\Diamond$ must be modified as follows:

  $$\nu_{\Diamond_a}(T_i) = \Big(\{\Diamond_a(B, u)\}, \{(u, \Diamond_a(B, u)), (\Diamond_a(B, u), u))\}, \{(\Diamond_a(B, u), B)\} \Big)$$

  for each $a \in \mathcal{I}$ and for all $\Diamond_a B \in \Phi_i(u)$: i.e. add the symmetric edge at the same time a new node is introduced, this respect the strict increase of node depth (condition (iv) of definition 11) [5]. The same kind of adjustement can be used for tense forms of layered logics (i.e. where converse $-a$ of each $a \in \mathcal{I}$ is allowed.
- The case of one permutation property like e.g.

  $$\forall x, y, z \colon \exists u \colon (R_1(x, y) \;\&\; R_2(y, z)) \to (R_2(x, u) \;\&\; R_1(u, z))$$

  corresponding to the axiom $\Box_2\Box_1 p \to \Box_1\Box_2 p$ can be handled by the same kind of adjustement of the $\Diamond_2$ rule; use the rule below which is essentially (we omit the case of empty nodes):

  If $(x_j, x_{j+1}) \in R_1$ for $0 \leq j < n$ and $\Diamond_2 B \in \Phi_i(x_n)$ then
  $\nu_{perm}(T_i) = \{$
  $\{H_j(\overrightarrow{x}) \colon 0 \leq j < n\},$
  $\{(x_j, H_j(\overrightarrow{x})) \colon 0 \leq j \leq n\} \cup \{(H_j(\overrightarrow{x}), H_{j+1}(\overrightarrow{x})) \colon 0 \leq j < n\},$
  $\{(H_n(\overrightarrow{x}), B)\}$ $\}$

[5] Of course, for logic KB on its own, our present method is not interesting since it is known to have a PSPACE satisfiability problem.

with a modified definition of δ such that depth along R_2 is "heavier" than along R_1, this ensures that in the rule, all $H_j(\overrightarrow{x})$ are deeper than x_j's and is harmless w.r.t. proposition 1 of the present paper. Formulated in this way, the above rule handled permutation while respecting the increase of nodes depth which is the main point on which rely our complexity result.

We have defined a new class of modal logics called *layered* because their models can be constructed layer by layer. These logics are specified by properties of their models which are thus called layered.

We proved they are decidable and that their satisfiability problem lies in NEXPTIME. This is the best possible lower bound since some of them are known to be complete for this complexity class.

To achieve this, we designed a sound, complete tableau calculus that permitted us to ensure the small model property of layered logics. With the help of various possibilities (modification of the $\Diamond$ rules, of the depth definition,. . .) we believe that this technical tool may extend to other logics and even to some transitive layered logics by introducing loop tests over whole subgraphs. This is subject of ongoing work.

References

1. Baldoni, M.: Normal Multimodal Logics With Interaction Axioms. In: Basin, D., D'Agostino, M., Gabbay, D.M., Matthews, S., Viganò, L. (eds.) Labelled Deduction. Applied Logic Series, vol. 17, pp. 33–53. Kluwer Academic Publisher, Boston (2000)
2. Chellas, B.F.: Modal Logic: an introduction. Cambridge Univ. Press, Cambridge (1980)
3. Castilho, M.A., del Farinas Cerro, L., Gasquet, O., Herzig, A.: Modal Tableaux with Propagation Rules and Structural Rules. Fundamenta Informaticae 32(3-4), 281–297 (1997)
4. del Fariñas Cerro, L., Gasquet, O.: A general framework for pattern-driven modal tableaux. Logic Journal of the IGPL 10(1), 51–84 (2002)
5. Del Fariñas Cerro, L., Gasquet, O.: Tableaux Based Decision Procedures for Modal Logics of Confluence and Density. Fundamenta Informaticae 40(4), 317–333 (1999)
6. Gabbay, D.M.: Selective filtration in modal logic. Theoria 30, 323–330 (1970)
7. Gasquet, O.: On the influence of confluence in modal logics. Fundamenta Informaticae, 70(3) (2005)
8. Goré, R.: Tableau methods for modal and temporal logics. In: D'Agostino, M., Gabbay, D., Hähnle, R., Posegga, J. (eds.) Handbook of Tableau Methods, Kluwer Academic Publishers, Boston (1999)
9. Ladner, R.: The computational complexity of provability in systems of modal logic. SIAM Journal on Computing 6, 467–480 (1977)
10. Lemmon, E. J., Scott, D. S.: An introduction to modal logic. Oxford, Blackwell, (1977)
11. Massacci, F.: Single step tableaux for modal logics: methodology, computations, algorithms. Journal of Automated Reasoning, vol. 24 (2000)
12. Shapirovsky, I.: On PSPACE-decidability in transitive modal logics. Proc. AiML'05, (2005)
13. Shethman, V.: Filtration via bisimulation. Proc. AiML'05 (2005)

The Neighbourhood of S0.9 and S1

Roderic A. Girle

Philosophy Department, University of Auckland
Auckland, New Zealand
r.girle@auckland.ac.nz

Abstract. Considerable work has been undertaken on the neighbourhood semantics for **S1** but little on **S0.9**. In this paper we use tableaux to represent the *model-set* / *model-system* neighbourhood semantics for both **S1** and **S0.9** and some other nearby systems.[1] **S0.9** is often seen as a more "natural" logic in the sequence: **S0.5** to **S0.9** to **S2** to **S3** than **S1**. This perception is discussed in terms of the interpretation of alternate worlds or model-sets.

Keywords: **S0.9**, **S1**, model-set semantics, tableaux, interpretation.

1 Introduction

In [3] it was shown that **S1** and **S0.9** are distinct modal systems. The proof was based on the use of *model-set* / *model-system* [6] semantics. The semantics in possible worlds form were adopted by several authors [2,3] as the semantics for **S1**. The semantics are rather strange It was shown by Cresswell [2] that they are actually the semantics for **S1+**, which includes but is not **S1**. Cresswell also showed, in standard possible worlds truth value semantics, that the semantics for **S1** is a *neighbourhood* semantics. Segerberg and Chellas [1] have, more recently, explored the strictly formal relationships between the several systems close to **S1**, including **S0.9**.

The purpose of this paper is to set out the model-set / model-systems semantics and their tableaux (truth trees) for systems in the vicinity **S0.9** and **S1**, and to comment about one way these systems are often seen in relationship to each other. It is sometimes suggested that there is a *natural flow* from **S0.5** to **S0.9** to **S2** to **S3**, a flow which leaves **S1** and **S1+** aside. We will comment later on this natural flow, and see how this either illuminates or obscures the scene.

2 Systems and Tableaux

In this paper we take it that tableaux are simply a diagrammatic way of representing the working of model-set / model-system semantics. This is in accord with Hintikka's use of model-set/model-system semantics to establish validity by means of *reductio* proofs. There are many other ways of taking tableau. They have been analysed in terms of graphs, and in terms of standard truth-value semantics. Not so here.

[1] The semantics for **S1** herein resulted from discussions with Graham Priest and Jerry Seligman in 2000.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 119–132, 2007.

Neighbourhood semantics have not readily converted to tableau systems. But, strange as the **S1+** semantics are, they readily yield a tableau system. The model-set / model-system semantics were set out in [3].

We begin by setting out the Lemmon axiom systems for **S0.5, S0.9, S1**, **S1+** and **S2**. It is also useful to set out **K**, even though we make only passing use of it. Tableau systems will then be set out for all six systems.

We use the standard language of propositional modal logic with ~, &, ⊃, $\Box$ and $\Diamond$. We define ↔ with:

Def ↔ $(A \leftrightarrow B) =_{df} (\Box(A \supset B) \,\&\, \Box(B \supset A))$

The axiom schema for **S0.5**, **S0.9**, **S1**, **S1+, S2** and **K** are drawn from:

A1. $(A \supset (B \supset A))$
A2. $((A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C)))$
A3. $((\sim A \supset \sim B) \supset (B \supset A))$
A4. $(\Box A \supset A)$
A5. $(\Box(A \supset B) \supset (\Box A \supset \Box B))$
A6. $(\Box(A \supset B) \supset (\Box(B \supset C) \supset \Box(A \supset C)))$
A7. $\Box((\Box A \,\&\, \Box B) \supset \Box(A \,\&\, B))$

We define the following axiom schema:

$\Box Ai$ ($1 \leq i \leq 7$) is the schema resulting from the prefixing of $\Box$ before Axiom Schema A*i*.

The Rules of Inference are:

R1. $\vdash A, \vdash (A \supset B) \Rightarrow \vdash B$ (***Modus Ponens***)
R2. $\vdash A \Rightarrow \vdash \Box A$ **(*Necessitation*)**
R3. $\vdash \Box A \Rightarrow \vdash A$ **(*De-necessitation*)**
R4. $\vdash (A \leftrightarrow B) \Rightarrow \vdash (\Box A \leftrightarrow \Box B)$
R5. $\vdash (A \supset B) \Rightarrow \vdash (\Box A \supset \Box B)$

The set of Axiom Schema and Inference Rules for Propositional Logic (**PL**) is:

PL = {A1, A2, A3, **R1**}

We also have:

S0.5 = {$\Box$A1, $\Box$A2, $\Box$A3, A4, A5, **R1**}
S0.9 = {$\Box$A1, $\Box$A2, $\Box$A3, $\Box$A4, $\Box$A5, **R1, R3, R4**}
S1 = {$\Box$A1, $\Box$A2, $\Box$A3, $\Box$A4, $\Box$A6, **R1**, **R3, R4**}
S1+ = {$\Box$A1, $\Box$A2, $\Box$A3, $\Box$A4, $\Box$A6, A7, **R1**, **R3, R4**}
S2 = {$\Box$A1, $\Box$A2, $\Box$A3, $\Box$A4, $\Box$A5, **R1**, **R3, R5**}
K = {A1, A2, A3, A5, **R1, R2**}

We begin with **S0.5** because the tableau system for **S0.5** is the simplest system for non-normal logics, and will be a good stepping-off point for the tableau systems for the more complex logics. But before we set out the tableau systems we set out the *model-set* / *model-system* semantics for all four systems.

An **S**-*model-system*, **Ω**, is a set of sets: *n, m, l, k, j, i, r*, ..., of formulas of **S**, which satisfies a set of consistency conditions. The members of any *model-system* are *model-sets*. The *model-sets* are the field of a binary relation, R, known as an alternate (or accessibility) relation. There is also a binary relation, N, between model sets and subsets of $\mathbf{\Omega}$, known as the *neighbourhood* relation.

A model-set, μ, to which some model-set, say ν, has the relation R, $(\exists\nu)\nu R\mu$, is an *alternate model-set.* A model-set, μ, to which no model-set has the relation R, $\sim(\exists\nu)\nu R\mu$, is a *non-alternate model-set.*

We define **S**-*Validity* (where **S** is one of the five above) in a “tableau friendly” way:

S-*Valid*(A) iff there is no *non-alternate model-set* $\mu \in \mathbf{\Omega}$ such that $\sim A \in \mu \in \mathbf{\Omega}$

We begin with the consistency conditions for classical propositional logic (**S** = **PL**):

(C. ∅)	No *model-set* contains both A and $\sim A$.
(C.~ ~)	If $\sim\sim A \in \mu \in \mathbf{\Omega}$ then $A \in \mu$.
(C. &)	If $(A \,\&\, B) \in \mu \in \mathbf{\Omega}$ then $A \in \mu$ and $B \in \mu$.
(C. ~ &)	If $\sim(A \,\&\, B) \in \mu \in \mathbf{\Omega}$ then either $\sim A \in \mu$ or $\sim B \in \mu$.
(C. ⊃)	If $(A \supset B) \in \mu \in \mathbf{\Omega}$ then either $\sim A \in \mu$ or $B \in \mu$.
(C. ~ ⊃)	If $\sim(A \supset B) \in \mu \in \mathbf{\Omega}$ then $A \in \mu$ and $\sim B \in \mu$.

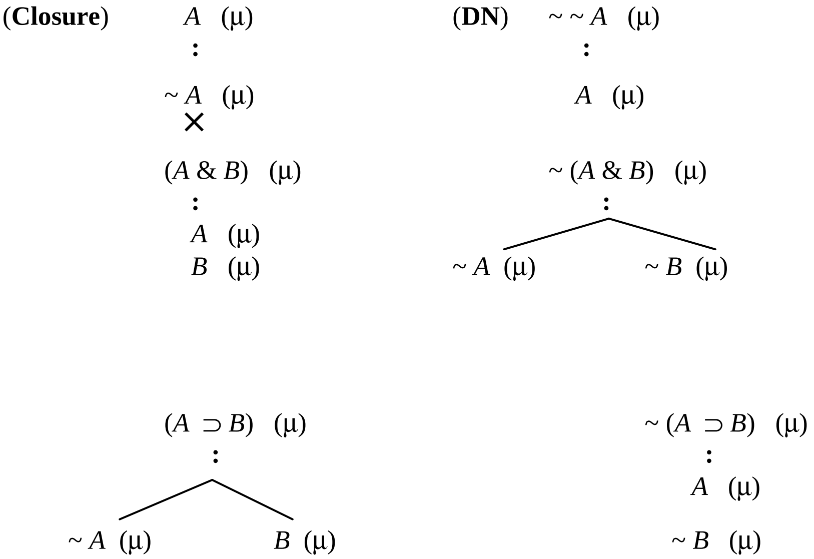
Let this set of consistency conditions be **PLC**:

PLC = {(C. ∅), (C.~ ~), (C. &), (C. ~ &), (C. ⊃), (C. ~ ⊃)}

These conditions translate directly into the following completely standard tableau rules for **PL**. They are in the style of [5], and in setting out the tableau rules we write:

“A (μ)” for “$A \in \mu$ ” where μ ranges across model-sets in any model-system.

Two standard rule names are also inserted.



Although it is not our aim to discuss the normal modal logics, it will assist in setting out of the non-normal modal logics if we set out the model-set/model-system semantics for **K**, usually taken to be the minimal normal modal logic.

The consistency conditions for **K** are **PLC** plus the following four conditions. The first two are for negated box and negated diamond.

(C. $\sim\Diamond$) If $\sim\Diamond A \in \mu \in \mathbf{\Omega}$ then $\Box \sim A \in \mu$.

(C. $\sim\Box$) If $\sim\Box A \in \mu \in \mathbf{\Omega}$ then $\Diamond \sim A \in \mu$.

These two conditions with the **DN** condition set out the standard interdefinability of box and diamond, and are common to all the modal logics we discuss. We let:

MN = {(C. $\sim\Diamond$), (C. $\sim\Box$)}

where "**MN**" is for "Modal Negation". The conditions in **MN** are represented by the following tableau rules:

$$
\begin{array}{llcll}
(\mathbf{MN}) & \sim\Diamond A \quad (\mu) & \qquad & (\mathbf{MN}) & \sim\Box A \quad (\mu) \\
 & \quad \vdots & & & \quad \vdots \\
 & \Box \sim A \quad (\mu) & & & \Diamond \sim A \quad (\mu)
\end{array}
$$

Now, for the **K** consistency conditions for positive box and diamond:

(C. $\Diamond$) If $\Diamond A \in \mu \in \mathbf{\Omega}$ then there is at least one model-set in $\mathbf{\Omega}$, say ν, such that ν is an alternate to μ, $\mu R\nu$, and $A \in \nu$.

(C. $\Box$) If $\Box A \in \mu \in \mathbf{\Omega}$ and ν is an alternate to μ, $\mu R\nu$, then $A \in \nu$.

These are the standard diamond and box consistency conditions for the normal modal logics. The set of consistency conditions for **K** is **KC**:

KC = **PLC** $\cup$ **MN** $\cup$ {(C. $\Diamond$), (C. $\Box$)}

The conditions (C. $\Diamond$) and (C. $\Box$) are represented by the following tableau rules:

$$
\begin{array}{lllcll}
(\mathbf{R.}\Diamond) & \Diamond A & (\mu) & \qquad & (\mathbf{R.}\Box) & \Box A \quad (\mu) \\
 & \quad \vdots & & & & \mu R\nu \\
 & \mu R\nu & & & & \quad \vdots \\
 & \quad A & (\nu) \;\; \textit{where } \nu \textit{ is NEW to} & & & A \quad (\nu) \\
 & & \qquad \textit{this path of the tree.} & & &
\end{array}
$$

There is a sense in which, in a tableau, (**R.**$\Diamond$) is a *model-set generation* rule. For tableaux it will be useful to refer to "*generated* model-sets", where ν is the generated model-set in the rule above. ν is generated by $\Diamond A$ in μ and ν is an alternate to μ.

We now set out the model-system consistency conditions for **S0.5**. They are **PLC** plus the **MN** rules and the following three conditions.

(C.◊0.5) If $\Diamond A \in \mu \in \mathbf{\Omega}$ and μ is not an alternate to any set in $\mathbf{\Omega}$, then there is at least one model-set in $\mathbf{\Omega}$, say ν, such that ν is an alternate to μ, $\mu R\nu$, and $A \in \nu$.

(C. □0.5) If $\Box A \in \mu \in \mathbf{\Omega}$ and μ is not an alternate to any set in $\mathbf{\Omega}$ and ν is an alternate to μ, $\mu R\nu$, then $A \in \nu$.

These two conditions are similar to the **K** conditions. The differences between the consistency conditions for **K** and **S0.5** are the qualifications that stipulate that the conditions for diamond and box apply only to non-alternate model-sets. There is an third condition which is similarly qualified:

(C. **T□**) If $\Box A \in \mu \in \mathbf{\Omega}$ and μ is not an alternate to any set in $\mathbf{\Omega}$, then $A \in \mu$.

The set of consistency conditions for **S0.5** is **S0.5C**:

S0.5C = **PLC** ∪ **MN** ∪ {(C. **T□**), (C. ◊), (C. □)}

In the model-set semantics the alternate relation, *R*, has no *explicit* properties such as reflexivity or transitivity. So, equivalent logical outcomes of such properties on *R* such as reflexivity have to depend on conditions such as (C. **T□**).

The **S0.5** conditions give the **PL** and **MN** tableau rules plus the following tableau rules:

(**R0.5◊**) *where* μ *is not an alternate to (not generated by) any model-set*

$\Diamond A$ (μ)
:

$\mu R\nu$

A (ν) *where* ν *is NEW to this path of the tree.*

(**R0.5□**) *where* μ *is not an alternate to (not generated by) any model-set*

$\Box A$ (μ)
$\mu R\nu$
:

A (ν)

(**T□**) *where* μ *is not an alternate to (not generated by) any model-set*

$\Box A$ (μ)
:

A (μ)

The distinction between normal and non-normal worlds in the standard possible worlds semantics is not mirrored in any exact way in the consistency conditions of model-set/model-system semantics. There is an apparently different distinction between two sets of model-sets in any model-system to that drawn in standard possible world semantics. In the model-set/model-system semantics there are the model-sets which are alternates to some model-set, and so *generated* by rules such as **R**◊; and model-sets which are not alternate to any model-set and hence, *not generated*. The only non-generated model-set in a tableau is the root model-set, the one with which the tableau begins. In **S0.5** the root model-set has tableau rules exactly the same as those for all model-sets in model-system semantics for **T** ([5] Chapter 3). In generated model-sets in **S0.5** the only rules which deal with formulas with either box or diamond or their negations for the main operator are the **MN** rules.

For **S1+** there are consistency conditions additional to those for **S0.5**. They are additional conditions for box and diamond which apply to generated or non-alternate model-sets. From this point we will use "alternate" for the longer "alternate to some model-set in **Ω**" The conditions are:

(C. **A** ◊) If ◊$A \in \mu \in$ **Ω** and μ is an alternate, then either $A \in \mu$ or there is at least one model-set in **Ω**, say ν, such that ν is an alternate to μ, μ*R*ν, and ~ $A \in \nu$.

(C. **A**□) If □ $A \in \mu \in$ **Ω** and μ is an alternate and ν is an alternate to μ, μ*R*ν, then ~ $A \in \nu$.

(C. **R**□) If □ $A \in \mu \in$ **Ω**, then $A \in \mu$.

The set of consistency conditions for **S1+** are **S1+C**:

S1+C = **S0.5C** ∪ {(C. **A** ◊), (C. **A**□), (C. **R**□)}

These rules are crucial to the validation of both A6 and A7. The tableau rules are extended with three new rules for alternate model-sets:

(**AR**◊) σ*R*μ *for some* σ
◊ *A* (μ)
:
μ*R*ν

A (μ) | ~ *A* (ν) *where* ν *is NEW to this path of the tree.*

(**AR**□) σ*R*μ *for some* σ
□ *A* (μ)
μ*R*ν
:
~ *A* (ν)

(**R**□) □ *A* (μ)
:
A (μ)

All tableau begin the test of the validity of *A* with ~ *A* (*n*), with the annotation "**NTF**" for "Negated Test Formula." We use *n, k, l, m, …*as indices for model-sets. In the rules above, μ and ν range over the model-set indices.

Consider the tableau for the **S1+** formula: □((□*A* & □*B*) ⊃ □(*A* & *B*))

1. ~ □((□*A* & □*B*) ⊃ □(*A* & *B*)) (*n*) **NTF**
2. ◊ ~((□*A* & □*B*) ⊃ □(*A* & *B*)) (*n*)
3. *nRk*
4. ~ ((□*A* & □*B*) ⊃ □(*A* & *B*)) (*k*)
5. (□*A* & □*B*) (*k*)

6.	$\sim\Box(A \,\&\, B)$ (k)		
7.	$\Diamond\sim(A \,\&\, B)$ (k)		
8.	$\Box A$ (k)		
9.	$\Box B$ (k)		
10.	kRm [Everything is as expected to this point. Then strange things begin to happen.]		
11.	$\sim(A \,\&\, B)$ (k)		$\sim\sim(A \,\&\, B)$ (m) 3, 7, **AR◊**
12.	A (k) 8, **R□**		$(A \,\&\, B)$ (m) 11, **DN**
13.	B (k) 9, **R□**		$\sim A$ (m) 8, 10, **AR□**
14			$\sim B$ (m) 9, 10, **AR□**
15.	$\sim A$ (k)	$\sim B$ (k) 11, **PL**	A (m) 12, **PL**
16	×	×	B (m) 12, **PL**
			×

We have **S1**+-*Valid* $(\Box((\Box A \,\&\, \Box B) \supset \Box(A \,\&\, B)))$

Next we turn to **S1** rather than **S0.9**, and return to **S0.9** later. For **S1** we add to the conditions for **S0.5** a different set to those added for **S1+.** This requires the introduction of a neighbourhood function, *N*, which relates single model-sets to sets of model-sets which contain certain formulas. We first define:

$$|A| =_{\text{df}} \{\mu \in \mathbf{\Omega} : A \in \mu\}$$

Segerberg and Chellas [1] point out that each $|A|$ is the *truth-set* of *A*.

Clearly $|A| \subseteq \mathbf{\Omega}$

We add to **S0.5** the following conditions for alternate model-sets:

(C. **S1◊**) If $\Diamond A \in \mu \in \mathbf{\Omega}$ and μ is an alternate, then either $A \in \mu$ or *not* $\mu N|\sim A|$.

(C. **S1□**) If $\Box A \in \mu \in \mathbf{\Omega}$ and μ is an alternate, then $A \in \mu$ and $\mu N|A|$.

(C. **Ω1**) If $\mu \in \mathbf{\Omega}$ and $\mu N|A|$ and $\mu N|B|$, then there is some model-set in **Ω**, say ν, such that $\sim A \in \nu$ and $\sim B \in \nu$

The condition (C. **Ω1**) is equivalent to the possible worlds stipulation in [2] that:

Prop 1: For all $\mu \in \mathbf{\Omega}$**:** if $\mu N|A|$ and $\mu N|B|$ then $|A| \cup |B| \neq \mathbf{\Omega}$

Clearly, if the union of $|A|$ and $|B|$ is *not* the whole model-system **Ω**, as stipulated by the principle above, then there has to be at least one model-set, say ν, which is not gathered into either $|A|$ or $|B|$ by the *N* relation, that is, both $\sim A$ and $\sim B$ are in that model-set, ν.

The set of consistency conditions for **S1** are **S1C**:

S1C = **S0.5C** $\cup$ {(C. **S1** ◊), (C. **S1□**), (C. **Ω1**)}

Tableau rules are:

(S1R◊)

$$\begin{array}{cc} \multicolumn{2}{c}{\sigma R\mu \quad \textit{for some } \sigma} \\ \multicolumn{2}{c}{\Diamond A \quad (\mu)} \\ \multicolumn{2}{c}{\vdots} \\ A \quad (\mu) & \textit{not } \mu N \mid {\sim} A| \end{array}$$

(S1R□)

$$\begin{array}{c} \sigma R\mu \quad \textit{for some } \sigma \\ \Box A \quad (\mu) \\ \vdots \\ \mu N \mid A| \end{array}$$

(R□)

$$\begin{array}{c} \Box A \quad (\mu) \\ \vdots \\ A \quad (\mu) \end{array}$$

(S Ω1)

$$\begin{array}{l} \mu N \mid A \mid \\ \mu N \mid B \mid \\ \quad \vdots \\ {\sim} A \quad (\nu) \\ {\sim} B \quad (\nu) \end{array}$$

where ν *is NEW to this path of the tree.*

We also need rules with which to manipulate neighbourhoods.

(**NN**)

$$\begin{array}{cc} \multicolumn{2}{c}{\mu N \mid A \mid} \\ \multicolumn{2}{c}{\textit{not } \mu N \mid B \mid} \\ \multicolumn{2}{c}{\vdots} \\ A \quad (\nu) & B \quad (\nu) \\ {\sim} B \quad (\nu) & {\sim} A \quad (\nu) \end{array}$$

where ν *is NEW to these paths of the tree.*

(**Closure**)

$$\begin{array}{c} \mu N \mid A \mid \\ \textit{not } \mu N \mid A \mid \\ \vdots \\ \times \end{array}$$

We also need a set of tableau rules, (**NPL**), by means of which truth-set equivalences can be dealt with:

$$\begin{array}{c} \mu N \mid {\sim}{\sim} A \mid \\ \vdots \\ \mu N \mid A \mid \end{array} \qquad \begin{array}{c} \mu N \mid {\sim}(A \supset B) \mid \\ \vdots \\ \mu N \mid A \mid \\ \mu N \mid {\sim} B \mid \end{array} \qquad \begin{array}{cc} \multicolumn{2}{c}{\mu N \mid (A \supset B) \mid} \\ \multicolumn{2}{c}{\vdots} \\ \mu N \mid {\sim} A \mid & \mu N \mid B \mid \end{array}$$

The introduction of the neighbourhood relation, *N*, takes **S1** into quite another realm. We look first at an **S1** tableau for the **S1** axiom:

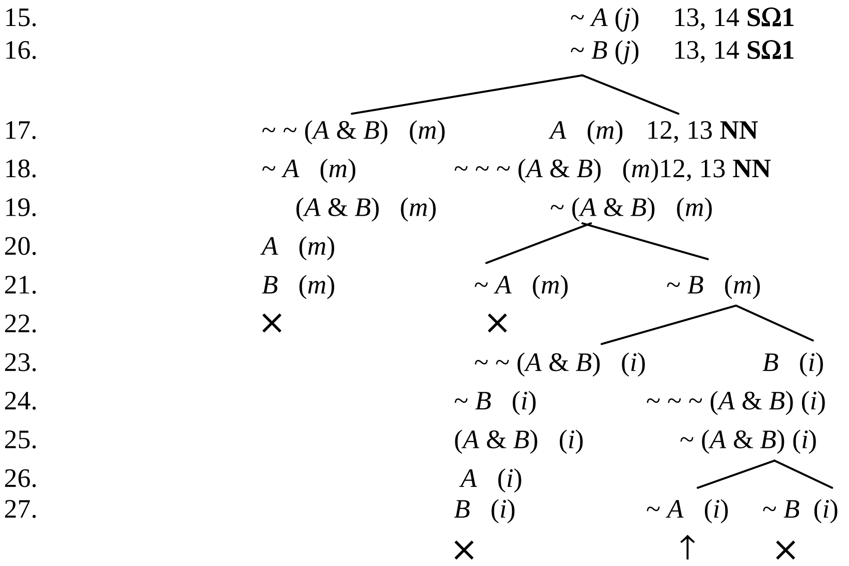
$\Box((\Box(A \supset B)\ \&\ \Box(B \supset C)) \supset \Box(A \supset C))$

1. $\sim \Box((\Box(A \supset B)\ \&\ \Box(B \supset C)) \supset \Box(A \supset C))$ (n) **NTF**
2. $\Diamond{\sim}((\Box(A \supset B)\ \&\ \Box(B \supset C)) \supset \Box(A \supset C))$ (n)

3.		*nRk*			
4.		~((□(*A* ⊃ *B*) & □(*B* ⊃ *C*)) ⊃ □(*A* ⊃ *C*)) (*k*)			
5.		(□ (*A* ⊃ *B*) & □(*B* ⊃ *C*)) (*k*)			
6.		~□(*A* ⊃ *C*) (*k*)			
7.		◊~ (*A* ⊃ *C*) (*k*)			
8.		□(*A* ⊃ *B*) (*k*)			
9.		□(*B* ⊃ *C*) (*k*)			
10.		(*A* ⊃ *B*) (*k*)			8, **R□**
11.		(*B* ⊃ *C*) (*k*)			9, **R□**
12.		~ (*A* ⊃ *C*) (*k*)		*not kN* | ~ ~ (*A* ⊃ *C*)|	7, **S1R◊**
13.		*A* (*k*)		*kN* | (*A* ⊃ *B*) |	8, **S1R□**
14.		~ *C* (*k*)		*kN* | (*B* ⊃ *C*) |	9, **S1R□**
15.				~ (*A* ⊃ *B*) (*j*)	13, 14 **SΩ1**
16.	~ *A* (*k*)	*B* (*k*)		~ (*B* ⊃ *C*) (*j*)	13, 14 **SΩ1**
17.	×			*A* (*j*)	15, **PL**
18.		~ *B* (*k*)	*C* (*k*)	~ *B* (*j*)	15, **PL**
19.		×	×	*B* (*j*)	16, **PL**
20.				~ *C* (*j*)	16, **PL**
21.				×	18, 19

We turn next to the Cresswell formula in [2] which shows that **S1+** is not the same as **S1**: **S1+** has none of the neighbourhood complications. Before we look at **S0.9**, we consider the use of the **S1** tableau system to disprove the **S1+** axiom.

1.	~□((□*A* & □*B*) ⊃ □(*A* & *B*)) (*n*)		**NTF**
2.	◊~ ((□*A* & □*B*) ⊃ □(*A* & *B*)) (*n*)		
3.	*nRk*		
4.	~ ((□*A* & □*B*) ⊃ □(*A* & *B*)) (*k*)		
5.	(□*A* & □*B*) (*k*)		
6.	~□(*A* & *B*) (*k*)		
7.	◊~ (*A* & *B*) (*k*)		
8.	□*A* (*k*)		
9.	□*B* (*k*)		
10.	*A* (*k*)		
11.	*B* (*k*)		
12.	~ (*A* & *B*) (*k*)	*not kN* | ~ ~ (*A* & *B*)|	3, 7, **S1R◊**
13.	× 12, 10, 11	*kN* | *A* |	8, **S1R□**
14.		*kN* | *B* |	9, **S1R□**


15. ~ A (j) 13, 14 SΩ1
16. ~ B (j) 13, 14 SΩ1
17. ~ ~ (A & B) (m) A (m) 12, 13 NN
18. ~ A (m) ~ ~ ~ (A & B) (m)12, 13 NN
19. (A & B) (m) ~ (A & B) (m)
20. A (m)
21. B (m) ~ A (m) ~ B (m)
22. × ×
23. ~ ~ (A & B) (i) B (i)
24. ~ B (i) ~ ~ ~ (A & B) (i)
25. (A & B) (i) ~ (A & B) (i)
26. A (i)
27. B (i) ~ A (i) ~ B (i)
× ↑ ×

This open tableau delivers an **S1** counter-example to the **S1**+-*Valid* formula. Let **S** be the set of model-sets in **Ω** which are not alternate. The counter-example is:

$\mathbf{\Omega} = \{n, k, j, m, i\}$ $\mathbf{S} = \{n\}$
$R = \{\langle n, k\rangle\}$ $N = \{\langle k, \{k, i\}\rangle, \langle k, \{k, m\}\rangle$

This is almost the same as the counter-example in Cresswell ([4] pg. 35).

Now for **S0.9**. In [3] the model-system semantics introduced a "relevance" condition for the **S0.9** system. We propose that (C. **Ω0.9**) be a qualified (C. **Ω1**):

(C. **Ω**0.9) If $\mu \in \mathbf{\Omega}$ and $\mu N \mid A \mid$ and $\mu N \mid B \mid$, and all the sub-formulas of A are sub-formulas of B or conversely, then there is some model-set in **Ω**, say ν, such that $\sim A \in \nu$ and $\sim B \in \nu$

We then have:

S0.9C = **S0.5C** ∪ {(C. **S1** ◊), (C. **S1**□), (C. **Ω0.9**)}

The tableau rules for **S0.9** are the same as for **S1** except that (**S Ω1**) is replaced with (**S Ω0.9**) where $AS(\alpha)$ is the set of atomic sub-formulas formulas of α.

(**S Ω0.9**)

$\mu N \mid A \mid$
$\mu N \mid B \mid$ *and either* $AS(A) \subseteq AS(B)$ *or* $AS(B) \subseteq AS(A)$

:
$\sim A\ (\nu)$
$\sim B\ (\nu)$
where ν *is NEW to*
this path of the tree.

If this rule were to be applied in the tableau for the **S1** axiom above, then the tableau would not close, because lines 15 and 16 could not follow from lines 13 and 14.

The relevant parts of the four lines are:

13.	$kN \mid (A \supset B) \mid$		
14.	$kN \mid (B \supset C) \mid$		
15.	$\sim (A \supset B)$	(j)	13, 14 **SΩ1**
16.	$\sim (B \supset C)$	(j)	13, 14 **SΩ1**

There can be no application of the sub-formula rule, (**S Ω0.9**), to 13 and 14.

Consider also an **S0.9** tableau for □A5: $\Box(\Box(A \supset B) \supset (\Box A \supset \Box B))$

1.	$\sim \Box(\Box(A \supset B) \supset (\Box A \supset \Box B))$		(n)	**NTF**
2.	$\Diamond \sim (\Box(A \supset B) \supset (\Box A \supset \Box B))$		(n)	
3.	nRk			
4.	$\sim (\Box(A \supset B) \supset (\Box A \supset \Box B))$		(k)	
5.	$\Box(A \supset B)$		(k)	
6.	$\sim (\Box A \supset \Box B)$		(k)	
7.	$\Box A$		(k)	
8.	$\sim \Box B$		(k)	
9.	$\Diamond \sim B$		(k)	
10.	$(A \supset B)$		(k)	5, **R**□
11.	A		(k)	7, **R**□
12.	$\sim B$	(k)	*not* $kN \mid \sim \sim B \mid$	3, 9 **S1R**◊
13.	×	10, 11, 12, **PL**	$kN \mid (A \supset B) \mid$	3, 5 **S1R**□
14.			$kN \mid A \mid$	3, 7 **S1R**□
15.			$kN \mid B \mid$	13, 14 **SΩ0.9**
16.			*not* $kN \mid B \mid$	12 **NPL**
			×	15, 16

It is obvious that the sub-formulas of A are sub-formulas of $(A \supset B)$ for the application of rule (**SΩ0.9**) at line 15 to lines 13 and 14.

Finally we turn to the model-system conditions for **S2**. The following are added to the conditions for **S0.5**. The equivalent of the following are to be found in [6].

(C.**S2**◊) If $\Diamond A \in \mu \in \mathbf{\Omega}$ and μ is alternate, and if $\Box B \in \mu$, for any B, then there is at least one model-set in $\mathbf{\Omega}$, say ν, such that ν is an alternate to μ, $\mu R \nu$, and $A \in \nu$.

(C. **S2**□) If $\Box A \in \mu \in \mathbf{\Omega}$ and μ is alternate and ν is an alternate to μ, $\mu R \nu$, then $A \in \nu$.

(C. **R**□) If $\Box A \in \mu \in \mathbf{\Omega}$, then $A \in \mu$.

The set of consistency conditions for **S2** are **S2C**:

S2C = **S0.5C** ∪ {(C. **S2** ◊), (C. **S2**□), (C. **R**□)}

The tableau rules are:

(**S2R◊**) $\sigma R\mu$ *for some* σ

$\Box B$ (μ)
$\Diamond A$ (μ)
:
$\mu R\nu$
A (ν)
where ν *is NEW to this path of the tree.*

(**S2R□**) $\sigma R\mu$ *for some* σ

$\Box A$ (μ)
$\mu R\nu$
:
A (ν)

(**R□**) $\Box A$ (μ)
:
A (μ)

3 Completeness

The proofs of completeness for these systems are generated simply by showing that the maximal consistent sets of formulas in the standard proofs are, in fact, model-sets with the requisite properties for the respective logics. For example, the completeness proof for **S1** would be simply parasitic on Cresswell’s original proof in [2] and on [1]. Rather than repeat what has already been done we turn to the issue of the *natural flow* from **S0.5** to **S2**.

4 Natural Flow

When one looks at both the axiomatics and semantics for the logics around **S0.9** and **S1**, there is a sense in which the natural sequence of logics is **S0.5**, **S0.9**, **S2, S3**, **S4** and **S5**. The sequence can be labeled the *Point Nine* sequence. The same sequence without **S0.9** can be labeled the *Point Five* sequence. Some might argue that an even more natural sequence is the **S0.5**, **S0.9**, **S2, T**, **S4** and **S5** sequence. But our focus is on the sequence up to **S2**.

From an axiomatic perspective, the change from **S0.5** to **S0.9** is made by the addition of **R3** and **R4** and the necessitation of the two non-necessitated axioms of **S0.5**. The change from **S0.9** to **S2** only requires the strengthening of **R4** to **R5**. In all of this, the basic underlying axioms remain the same. Both the Point Nine, Point Five and T sequences have this simple flow.

S1 and **S1+** are to one side in the sense that axiom **A6** appears for both and then vanishes from the scene. It is not valid for **S0.9**, although it is for **S2**.

From a semantic perspective, the flow is not so smooth. **S0.9** is out of place because of the neighbourhood semantics. Neighbourhood semantics are not required for the rest of the Point Nine sequence. Otherwise, in the Point Five sequence, the shift up from **S0.5** to **S2**, and then on through the **S2** to **S5** systems is simple and incremental without any neighbourhood semantics. This is not to deny that trivial neighbourhood semantics can be constructed. But their triviality reinforces the point about the flow. All of **S0.9**, **S1** and **S1+** are to one side of the intuitively natural sequence of semantics.

An important issue that lurks behind all this concerns the interpretation of modal semantics. For example, it is easy to interpret **S5** semantics as the semantics of logical possibility so that “$\Diamond A$” is read as “Given what must be true, it’s logically possible that *A*”. Hintikka [7] has shown that **S4** can be interpreted by taking $\Diamond A$ to mean “Given what’s known, it’s possible that *A*” So there is an epistemic reading of **S4** from a semantic point of view. **S4** and **S5** are normal modal logics, so the same interpretation can be given in all worlds. But what can be said of the semantics of **S0.5**? Lemmon [8] made his suggestion that **S0.5** was best seen as an epistemic logic. In [4] **S0.5** is described as a Cartesian epistemic logic on the basis of the axiomatics alone. But what is to be said about its semantics? In alternate worlds knowledge and ignorance mean nothing. Although “$\Diamond \Box A$” might be read as “Given what’s known, it’s possible that *A* is known,” “*A* is known,” *in the context given*, has no epistemic interpretation from a semantic point of view. This can be seen from a fragment of a tableau for **S0.5** in which $\Diamond \Box A$ occurs in the root of the tableau:-

$$\begin{array}{ll} \Diamond \Box A & (n) \\ nRk & \\ \Box A & (k) \end{array}$$

Since *k* is an alternate world, there is no logic for the box in $\Box A$. It is as if $\Box A$ is treated as an atomic formula.

And that’s the focus of the difficulties with the interpretation of **S1**. How should the *semantic structures* of model-sets (or worlds), and in particular, the alternate model-sets or non-normal worlds, be interpreted?

Tableaux are systems for building a partial model or counter-model. We can take the root of the tableau to be a (partial) description of the actual world and the generated sets or world’s descriptions to be the non-actual possibilities. The normal modal logics give us a univocal or unchanging account of possibility across all worlds. In the non-normal logics the generated modal structures are governed by conditions which make it clear that the notions of possibility and necessity, let alone knowledge and ignorance, are not the same in the generated worlds as they are in actual world.

What kinds of differences might we expect in the alternate worlds? There is a spectrum of difference in one dimension with modality having no logical weight at all at one end through various qualifications to being the same as the actual world at the other end. But, what sorts of qualifications might there be?

Modality in **S0.5** generated worlds is at the no-logical-weight end of the spectrum. There is no modality apart from the interdefinability of box and diamond. And even that can be disposed of by having just diamond, or just box. In **S2**, modality operates only if there is both positive and negative modality. There are possibilities only if there are impossibilities (necessities). In **S3**, there is a situation similar to **S2**. So, for **S0.5**, **S2** and **S3** the tableau semantics indicate that we have what amounts to differences in the notion of possibility as we move further from the actual world and into worlds generated by possibility.

What can be said for **S0.9**, **S1** and **S1+**? Let us begin with the central system, **S1**. The tableau for **S1** indicate that there is, as with **S0.5**, **S2** and **S3**, a difference in the notion of possibility in the generated worlds from the actual world. The difference is quite complex. Cresswell says that the semantics for **S1** are “not particularly elegant” and that **S1** is “a very silly system.” ([2] page 6) Is Cresswell’s assessment correct, in spirit if not literally?

There are two elements at work in the non-normal systems. First is the valuation function. Second is the division between alternate and non-alternate, parallel to the normal and non-normal worlds. If we think of the Point Five sequence, then we see that the valuation function remains the same across all model sets. If set generation is allowed, then $\Diamond A$ generates a set in which there is A. In terms of possible worlds, if $\Diamond A$ is true, the there is a world in which A is true. The big change in **S1** comes with the original conditions (C.**A**◊) and (C.**A**□) from [3]. $\Diamond A$ generates a set in which there is $\sim A$. In terms of possible worlds, if $\Diamond A$ is true, the there is a world in which A is *false*, which led to the idea that box in alternate worlds changed its sense to the negative of box, to impossibility in alternate worlds. This is comprehensible even though this is a radical change. But these are the conditions applicable to **S1+** and not to **S1**. **S1+** does not require neighbourhood semantics. There is a valuation function switch within standard non-normal semantics. The actual conditions for **S1**, (C.**S1**◊) and (C.**S1**□), are far more perplexing. But that is not all.

In the case of **S1**, **Prop 1** is an issue. It is very difficult to see what this might be taken to mean. It just looks like an opportunistic invention which will give completeness and soundness. This might be very silly, but more like it shows what Sylvan [10] claimed – that any system could be provided with a formal semantics if we are inventive enough. But having a relational semantics may well tell us about inventiveness but nothing useful about meaning.

References

[1] Chellas, Brian, F., Segerberg, Krister.: Modal Logics in the Vicinity of S1. Notre Dame. Journal of Formal Logic 37(1), 1–24 (1996)

[2] Cresswell, Maxwell, J.: S1 is not so simple. In: Sinnott-Armstrong, W., Raffman, D., Asher, N. (eds.) Modality, Morality and Belief: Essays in Honour of Ruth Barcan Marcus, Cambridge University Press, Cambridge pp. 29–40

[3] Girle, Roderic, A.: S1 ≠ S0.9. Notre Dame. Journal of Formal Logic XVI (3), 339–344 (1975)

[4] Girle, Roderic, A.: Logical Fiction: Real vs. Ideal. In: Hing-Yan, L. (ed.) Pacific Rim International Conference on Artificial Intelligence, Singapore, Nov 22-27 (1998)

[5] Girle, Rod.: Modal Logics and Philosophy, Acumen, London (2000)

[6] Hintikka, J.J.K.: Modality and Quantification. Theoria 27, 119–128 (1961)

[7] Hintikka, J.J.K.: Knowledge and Belief. Cornell University Press, Ithaca (1962)

[8] Lemmon, E.J.: Is there only one correct system of modal logic? Aristotelian Society Supplementary XXXIII, 23–40 (1959)

[9] Scotch, Peter, K.: Remarks on the semantics of non-normal logics. Topoi 3, 85–90 (1984)

[10] Sylvan, Richard.: Relational Semantics for all Lewis, Lemmon and Feys modal Logics, and most notably for systems between S0.3° and S1. The Journal of Non.-classical Logic 6(99), 19–40 (1989)

EXPTIME Tableaux with Global Caching for Description Logics with Transitive Roles, Inverse Roles and Role Hierarchies

Rajeev Goré[1,⋆] and Linh Anh Nguyen[2]

[1] The Australian National University and NICTA
Canberra ACT 0200, Australia
Rajeev.Gore@anu.edu.au

[2] Institute of Informatics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
nguyen@mimuw.edu.pl

Abstract. The description logic $\mathcal{SHI}$ extends the basic description logic $\mathcal{ALC}$ with transitive roles, role hierarchies and inverse roles. The known tableau-based decision procedure [9] for $\mathcal{SHI}$ exhibit (at least) NEXPTIME behaviour even though $\mathcal{SHI}$ is known to be EXPTIME-complete. The automata-based algorithms for $\mathcal{SHI}$ often yield optimal worst-case complexity results, but do not behave well in practice since good optimisations for them have yet to be found. We extend our method for global caching in $\mathcal{ALC}$ to $\mathcal{SHI}$ by adding analytic cut rules, thereby giving the first EXPTIME tableau-based decision procedure for $\mathcal{SHI}$, and showing one way to incorporate global caching and inverse roles.

1 Introduction and Motivation

Description logics (DLs) are notational variants of multi-modal logics which have proved to be important in representing and reasoning about knowledge. We assume the reader is familiar with the notions of transitive roles, inverse roles, role hierarchies and TBoxes (global assumptions) in DLs [1].

The decision problem for most of these logics (with global assumptions) is EXPTIME-hard. The known decision procedures for these logics are tableau-based, resolution-based or automata-based. In practice, the automata-based methods do not behave as well as the other methods since good optimisations for them have not been found to date. For very expressive description logics, the most successful practical methods are all tableau-based.

Global assumptions or transitive roles can cause the traditional tableau methods for these logics to contain infinite branches because certain nodes repeat *ad infinitum*. The usual solution is to remember certain nodes created along the current branch using "histories" and to "block" all rules that would re-create an

[⋆] National ICT Australia is funded by the Australian Government's Dept of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence program.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 133–148, 2007.

exact copy of any remembered nodes [1,7]. For inverse roles, blocked nodes can become unblocked, and reblocked later, requiring "dynamic blocking" [9].

The basic description logic $\mathcal{ALC}$ extended with transitive roles, inverse roles, and role hierarchies is called $\mathcal{SHI}$. Horrocks and Sattler [9] gave a NEXPTIME tableau decision procedure for $\mathcal{SHI}$. In [1], Baader and Sattler wrote: *"The point in designing these algorithms* [for $\mathcal{SHI}$] *was not to prove worst-case complexity results, but $\cdots$ to obtain 'practical' algorithms $\cdots$ that are easy to implement and optimise, and which behave well on realistic knowledge bases. Nevertheless, the fact that 'natural' tableau algorithms for such EXPTIME-complete logics are usually NEXPTIME-algorithms is an unpleasant phenomenon. $\cdots$ Attempts to design EXPTIME-tableaux for such logics (De Giacomo et al., 1996; De Giacomo and Massacci, 1996; Donini and Massacci, 1999) usually lead to rather complicated (and thus not easy to implement) algorithms, which (to the best of our knowledge) have not been implemented yet."* [1, page 26].

More precisely, the mentioned work of De Giacomo and Massacci [2] gives only a NEXPTIME tableau decision procedure for propositional dynamic logic with converse (CPDL) and an informally described transformation of NEXPTIME tableaux into an EXPTIME decision procedure for CPDL using the "look behind analytic cut". Using analytic cuts to eliminate nondeterminism and reduce the complexity from NEXPTIME to EXPTIME is a good idea. But it is not clear how to eliminate the nondeterminism (of the NEXPTIME decision procedure for CPDL [2]) using "look behind analytic cut". Specifically, suppose we apply an or-rule to a node w and obtain two successor nodes u and v, and after that we follow the u-branch. If we later "look behind" and add a formula φ to w, then φ affects node v. But how to eliminate the or-branching at w ?

Thus, to the best of our knowledge, no "natural and easy to implement" EXPTIME tableau decision procedures have been given or implemented for $\mathcal{SHI}$. Here, we give such a procedure using analytic cuts to "guess the future" so we *never look behind*, in unison with our method of sound global caching for $\mathcal{ALC}$ [6].

Caching is a crucial optimisation technique for obtaining efficient tableau-based decision procedures for description logics [8,3]. Naive use of caching can be unsound, so complex and difficult to implement methods have been devised to regain EXPTIME decision procedures for the basic logic $\mathcal{ALC}$ [4]. But the decision procedure given by Donini and Massacci [4] for $\mathcal{ALC}$ permanently caches *"all and only unsatisfiable sets of concepts"*, and temporarily caches visited nodes on the current branch, even though this means that *"many potentially satisfiable sets of concepts are discarded when passing from a branch to another branch"*. Even more complicated methods have been suggested for handling inverse roles and transitive roles, but these require severe restrictions to retain soundness [3]. Our global caching method is based on building a simple and-or graph using a sound and complete traditional tableau calculus for $\mathcal{ALC}/\mathcal{SHI}$.

Our method for transforming our tableau calculus for $\mathcal{SHI}$ into an EXPTIME decision procedure for $\mathcal{SHI}$ is almost the same as that for $\mathcal{ALC}$ in our previous work [6]. The minor differences are some alterations for $\mathcal{SHI}$ and that some of the tableau rules for $\mathcal{SHI}$ carry their principal concepts into their denominators

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \qquad \bot^{\mathcal{I}} = \emptyset \qquad (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \qquad (r^-)^{\mathcal{I}} = (r^{\mathcal{I}})^{-1}$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$(C \sqsubseteq D)^{\mathcal{I}} = (\neg C \sqcup D)^{\mathcal{I}} \qquad (C \doteq D)^{\mathcal{I}} = ((C \sqsubseteq D) \sqcap (D \sqsubseteq C))^{\mathcal{I}}$$
$$(\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \text{ implies } b \in C^{\mathcal{I}}\}$$
$$(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \text{ and } \ b \in C^{\mathcal{I}}\}$$

Fig. 1. Interpretation of Concepts and Roles

(which we need for combining the proofs of completeness of the calculus with correctness of global caching). The overlap is needed to make this paper self-contained. The major differences of this paper w.r.t. [6] lie in the use of the cut rules, and the non-trivial complications necessary to ensure that the resulting calculus and the algorithm resulting from it are complete.

In Section 2, we recall the notation and semantics of $\mathcal{SHI}$. In Section 3, we present our tableau calculus for $\mathcal{SHI}$ and prove its soundness, and in Section 4, we prove its completeness. In Section 5, we present a simple EXPTIME decision procedure for $\mathcal{SHI}$ that is based on the calculus and uses global caching and propagation. In Section 6, we outline further work and conclusions.

2 Notation and Semantics of $\mathcal{SHI}$

Given a finite set RNames of role names, let $\text{RNames}^- = \{r^- \mid r \in \text{RNames}\}$ be the set of *inverse roles*. A *$\mathcal{SHI}$ role* is any member of $\text{RNames} \cup \text{RNames}^-$. We use uppercase letters like R and S for $\mathcal{SHI}$ roles. For any $\mathcal{SHI}$ role R the inverse role $R^- = r^-$ if $R = r$ and $R^- = r$ if $R = r^-$.

We use A for an atomic concept and use C and D for arbitrary concepts. Concepts in $\mathcal{SHI}$ are formed using the following BNF grammar:

$$C, D ::= \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid C \sqsubseteq D \mid C \doteq D \mid \forall R.C \mid \exists R.C$$

An *interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ consists of a non-empty set $\Delta^{\mathcal{I}}$, the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$, the *interpretation function* of $\mathcal{I}$, that maps every atomic concept to a subset of $\Delta^{\mathcal{I}}$ and every role name to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to interpret all $\mathcal{SHI}$ concepts and roles as shown in Figure 1. An interpretation $\mathcal{I}$ *satisfies* a concept C if $C^{\mathcal{I}} \neq \emptyset$, and *validates* C if $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$. Clearly, $\mathcal{I}$ *validates* C iff it does not *satisfy* $\neg C$.

A TBox (of global axioms/assumptions) $\mathcal{T}$ is a finite set of concepts. (Traditionally, $C \sqsubseteq D$ and $C \doteq D$ are not treated as concept constructors, and a TBox is defined to be a finite set of terminological axioms of the form $C \sqsubseteq D$ or $C \doteq D$, where C and D are concepts, but the two definitions are equivalent.) An interpretation $\mathcal{I}$ is a *model* of $\mathcal{T}$ if $\mathcal{I}$ validates all concepts in $\mathcal{T}$.

A *$\mathcal{SHI}$ RBox* $\mathcal{R}$ is a finite set of role inclusion axioms $R \sqsubseteq S$ and transitivity axioms $R \circ R \sqsubseteq R$ for $\mathcal{SHI}$ roles R and S satisfying:

RBox 1: $R \sqsubseteq R \in \mathcal{R}$ for every $\mathcal{SHI}$-role R,
RBox 2: $R \sqsubseteq S \in \mathcal{R}$ implies $R^- \sqsubseteq S^- \in \mathcal{R}$,
RBox 3: $R \circ R \sqsubseteq R \in \mathcal{R}$ implies $R^- \circ R^- \sqsubseteq R^- \in \mathcal{R}$,
RBox 4: $R \sqsubseteq R' \in \mathcal{R}$ and $R' \sqsubseteq R'' \in \mathcal{R}$ imply $R \sqsubseteq R'' \in \mathcal{R}$, and
RBox 5: $R \sqsubseteq S \in \mathcal{R}$ and $S \sqsubseteq R \in \mathcal{R}$ imply $R = S$.

An interpretation $\mathcal{I}$ is a *model* of $\mathcal{R}$ if $\mathcal{I}$ validates all axioms in $\mathcal{T}$: that is, $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ if $R \sqsubseteq S \in \mathcal{R}$, and $R^{\mathcal{I}} \circ R^{\mathcal{I}} \subseteq R^{\mathcal{I}}$ if $R \circ R \sqsubseteq R \in \mathcal{R}$.

We use X, Y for finite sets of concepts. We say that $\mathcal{I}$ *satisfies* X if there exists $d \in \Delta^{\mathcal{I}}$ such that $d \in C^{\mathcal{I}}$ for all $C \in X$. Note: by our definition, satisfaction is defined "locally", and $\mathcal{I}$ satisfies X does not mean that $\mathcal{I}$ is a model of X.

We say that $(\mathcal{R},\mathcal{T})$ *entails* C, and write $(\mathcal{R},\mathcal{T}) \models C$, if every model of $\mathcal{R}$ and $\mathcal{T}$ validates C. We say that C is satisfiable w.r.t. $(\mathcal{R},\mathcal{T})$ if some model of $\mathcal{R}$ and $\mathcal{T}$ satisfies $\{C\}$. Similarly, X is satisfiable w.r.t. $(\mathcal{R},\mathcal{T})$ if some model of $\mathcal{R}$ and $\mathcal{T}$ satisfies X. Observe that $(\mathcal{R},\mathcal{T}) \models C$ iff $\neg C$ is unsatisfiable w.r.t. $(\mathcal{R},\mathcal{T})$.

For "tableaux" defined in the next section we use also the concept constructor $\forall^\dagger R.C$, where C is a concept without $\forall^\dagger$. A concept $\forall^\dagger R.C$ has the same semantics as $\forall R.C$ (i.e. $(\forall^\dagger R.C)^{\mathcal{I}} = (\forall R.C)^{\mathcal{I}}$) but its use in tableaux is more restricted due to $\widetilde{Sf}(\_)$ defined as follows. If C is not of the form $\forall^\dagger R.D$ then let $\widetilde{Sf}(C)$ be the set of all sub-concepts of C and sub-concepts of $\overline{C}$ including themselves, else let $\widetilde{Sf}(C) = \widetilde{Sf}(D)$. Let $\widetilde{Sf}(X) = \bigcup_{C \in X} \widetilde{Sf}(C) \cup \{\bot\}$ and let $\widetilde{Sf}_{\mathcal{R}}(X)$ be the set $\widetilde{Sf}(X) \cup \{\forall S.C, \exists S.\overline{C} \mid (\forall R.C) \in \widetilde{Sf}(X)$ and $S \sqsubseteq R \in \mathcal{R}$ for some $R\}$.

Note: We now assume that concepts are in *negation normal form*, where $\doteq$ and $\sqsubseteq$ are translated away and $\neg$ occurs only directly before atomic concepts, treating $\forall^\dagger$ as $\forall$. In $\mathcal{SHI}$, every concept C has a logically equivalent concept C' which is in negation normal form. We write $\overline{C}$ for the negation normal form of $\neg C$: thus, $\overline{\forall^\dagger R.D} = \overline{\forall R.D} = \exists R.\overline{D}$ and $\overline{\exists R.D} = \forall R.\overline{D}$.

3 A Tableau Calculus for $\mathcal{SHI}$

We consider tableaux w.r.t. $(\mathcal{R},\mathcal{T})$, where $\mathcal{R}$ is a $\mathcal{SHI}$ RBox and $\mathcal{T}$ is a TBox. A *tableau rule* w.r.t. $(\mathcal{R},\mathcal{T})$ consists of a numerator X and a (finite) list of denominators $Y_1, Y_2, \ldots, Y_k$, all of which are finite sets of concepts. Such a rule (ρ) is written in the form

$$(\rho)\ \frac{X}{Y_1 \mid \ldots \mid Y_k}$$

As we shall see later, each rule is read downwards as "if the numerator X is satisfiable w.r.t. $(\mathcal{R},\mathcal{T})$, then there exists a denominator Y_i which is also satisfiable w.r.t. $(\mathcal{R},\mathcal{T})$". The numerator of each tableau rule contains one or more distinguished concepts called the *principal concepts*.

We write $X;Y$ for $X \cup Y$, and $X;C$ for $X \cup \{C\}$.

The tableau calculus $\mathcal{CSHI}$ for $\mathcal{SHI}$ is shown in Figure 2. The rules $(\sqcap)$, $(\sqcup)$, (H), $(H^\dagger)$, $(cut_\forall)$, (cut_B), and (cut_5) are *static* rules, while $(\exists R)$ is a *transitional rule*. Note that we include the principal concept of the static rules in their denominators. Thus, the numerator of any static rule is a subset of every one of

$$(\bot)\ \frac{X\ ;\ C\ ;\ \overline{C}}{\bot} \qquad (\sqcap)\ \frac{X\ ;\ C \sqcap D}{X\ ;\ C \sqcap D\ ;\ C\ ;\ D} \qquad (\sqcup)\ \frac{X\ ;\ C \sqcup D}{X\ ;\ C \sqcup D\ ;\ C \mid X\ ;\ C \sqcup D\ ;\ D}$$

$$(H)\ \frac{X\ ;\ \forall R.C}{X\ ;\ \forall R.C\ ;\ \forall S.C}\ \text{ if } S \sqsubseteq R \in \mathcal{R} \qquad (H^\dagger)\ \frac{X\ ;\ \forall^\dagger R.C}{X\ ;\ \forall^\dagger R.C\ ;\ \forall^\dagger S.C}\ \text{ if } S \sqsubseteq R \in \mathcal{R}$$

$$(cut_\forall)\ \frac{X}{X\ ;\ \forall R.C \mid X\ ;\ \exists R.\overline{C}}\quad \text{if } \forall R.C \in \widetilde{Sf}_{\mathcal{R}}(X \cup \mathcal{T})$$

$$(cut_B)\ \frac{X}{X\ ;\ C \mid X\ ;\ \overline{C}\ ;\ \forall^\dagger R^-.\exists R.\overline{C}}\quad \text{if } \forall R.C \in \widetilde{Sf}_{\mathcal{R}}(X \cup \mathcal{T})$$

$$(cut_5)\ \frac{X}{X\ ;\ \forall R.C \mid X\ ;\ \exists R.\overline{C}\ ;\ \forall^\dagger R^-.\exists R.\overline{C}}\quad \begin{array}{l}\text{if } \forall R.C \in \widetilde{Sf}_{\mathcal{R}}(X \cup \mathcal{T})\\ \text{and } R \circ R \sqsubseteq R \in \mathcal{R}\end{array}$$

$$(\exists R)\ \frac{X\ ;\ \exists R.C}{trans_{\mathcal{R}}(X,R)\ ;\ C\ ;\ \mathcal{T}}\ \text{ where}$$

$$\begin{aligned} trans_{\mathcal{R}}(X,R) = {} & \{D \mid ((\forall S.D \in X) \lor (\forall^\dagger S.D \in X)) \land (R \sqsubseteq S \in \mathcal{R})\} \\ & \cup \{\forall S.D \in X \mid (R \sqsubseteq S \in \mathcal{R}) \land (S \circ S \sqsubseteq S \in \mathcal{R})\} \\ & \cup \{\forall^\dagger S.D \in X \mid (R \sqsubseteq S \in \mathcal{R}) \land (S \circ S \sqsubseteq S \in \mathcal{R})\} \end{aligned}$$

Fig. 2. Tableau Rules w.r.t. $(\mathcal{R}, \mathcal{T})$ for Calculus $\mathcal{CSHI}$

its denominators. The subscripts in (cut_B) and (cut_5) are names of the modal axioms $(B) : \varphi \to \Box\Diamond\varphi$ and $(5) : \Diamond\varphi \to \Box\Diamond\varphi$, from which they are derived.

A set X is *closed* w.r.t. a tableau rule if applying that rule to X gives back X as one of the denominators. *We implicitly assume that a static rule is applied to X only when X is not closed w.r.t. that rule and treat this as an (additional) condition for applying the rule.* Consequently, the rules (H) and $(H^\dagger)$ require that $S \neq R$ and the static rules with two denominators are applicable only if *both denominators are different from their numerator.*

A *$\mathcal{CSHI}$-tableau* (tableau, for short) for X w.r.t. $(\mathcal{R}, \mathcal{T})$ is a tree with root $(X; \mathcal{T})$ whose nodes carry finite sets of concepts obtained from their parent nodes by instantiating a $\mathcal{CSHI}$-tableau rule w.r.t. $(\mathcal{R}, \mathcal{T})$ with the proviso that: if a child s carries a set Y and no rule is applicable to Y or Y has already appeared on the branch from the root to s then s is an *end node*.

Remark 1. Note that the rules guarantee that $\mathcal{T}$ is contained in every world of the model under construction to ensure that $\mathcal{T}$ is globally satisfied as required, and to allow a simpler presentation of the rules.

A branch in a tableau is *closed* if its end node carries only $\bot$. A tableau is *closed* if every one of its branches is closed. A tableau is *open* if it is not closed.

A finite set X of concepts is *consistent* w.r.t. $(\mathcal{R}, \mathcal{T})$ if every tableau for X w.r.t. $(\mathcal{R}, \mathcal{T})$ is open. If some tableau for X w.r.t. $(\mathcal{R}, \mathcal{T})$ is closed then X is *inconsistent* w.r.t. $(\mathcal{R}, \mathcal{T})$.

Calculus $\mathcal{CSHI}$ is *sound* if for every $\mathcal{SHI}$ RBox $\mathcal{R}$, every TBox $\mathcal{T}$, and every finite set X of concepts, X is satisfiable w.r.t. $(\mathcal{R}, \mathcal{T})$ implies X is consistent w.r.t. $(\mathcal{R}, \mathcal{T})$. It is *complete* if for every $\mathcal{SHI}$ RBox $\mathcal{R}$, every TBox $\mathcal{T}$, and every finite set X of concepts, X is consistent w.r.t. $(\mathcal{R}, \mathcal{T})$ implies X is satisfiable w.r.t. $(\mathcal{R}, \mathcal{T})$.

Example 1. Let $\mathcal{R}$ be the smallest RBox satisfying the conditions adopted for RBoxes and containing $S \circ S \sqsubseteq S$ and $S \sqsubseteq R$. We give below a closed $\mathcal{CSHI}$ tableau for $C \sqcap \exists S.\exists S.\forall R^-.\overline{C}$ w.r.t. $(\mathcal{R}, \emptyset)$, in which a superscript $*$ marks the principal concept of a rule application except the cut rules. As shown later, $\mathcal{CSHI}$ is sound, hence every model of $\mathcal{R}$ validates $C \sqsubseteq \forall S.\forall S.\exists R^-.C$.

$$\dfrac{C \sqcap^* \exists S.\exists S.\forall R^-.\overline{C}}{\dfrac{C;\ \exists S.\exists S.\forall R^-.\overline{C}}{\dfrac{C;\ \exists S.\exists S.\forall R^-.\overline{C}; \overline{C}}{\bot} \quad \Bigg| \quad \dfrac{C;\ \exists S.\exists S.\forall R^-.\overline{C};\ C;\ \forall^{\dagger^*} R.\exists R^-.C}{\dfrac{C;\ \exists^* S.\exists S.\forall R^-.\overline{C};\ C;\ \forall^\dagger R.\exists R^-.C;\ \forall^\dagger S.\exists R^-.C}{\dfrac{\exists^* S.\forall R^-.\overline{C};\ \exists R^-.C;\ \forall^\dagger S.\exists R^-.C}{\dfrac{\forall R^-.\overline{C};\ \exists R^-.C;\ \forall^\dagger S.\exists R^-.C}{\bot}}}}}}$$

A tableau rule w.r.t. $(\mathcal{R}, \mathcal{T})$ is *sound* if it has the property that if the numerator is satisfiable w.r.t. $(\mathcal{R}, \mathcal{T})$ then so is one of the denominators.

Lemma 1. *The calculus $\mathcal{CSHI}$ is sound.*

Proof. The calculus $\mathcal{CSHI}$ is sound because all rules of $\mathcal{CSHI}$ are sound.

Let the closure $\mathrm{Cl}(\mathcal{R}, \mathcal{T}, X)$ be the set

$$\{D, \forall^\dagger R.D \mid D \in \widetilde{Sf}_{\mathcal{R}}(X \cup \mathcal{T}) \text{ and } R \text{ is a } \mathcal{SHI}\text{-role appearing in } (\mathcal{R}, \mathcal{T}, X)\}.$$

Let n be the size of $(\mathcal{R}, \mathcal{T}, X)$, i.e. the sum of the lengths of the concepts of $X \cup \mathcal{T}$ and the lengths of the assertions of $\mathcal{R}$. Then the size of $\widetilde{Sf}_{\mathcal{R}}(X \cup \mathcal{T})$ is $O(n^2)$ and the size of $\mathrm{Cl}(\mathcal{R}, \mathcal{T}, X)$ is $O(n^3)$. Note that if a concept C appears in a tableau for X w.r.t. $(\mathcal{R}, \mathcal{T})$ then $C \in \mathrm{Cl}(\mathcal{R}, \mathcal{T}, X)$: calculus $\mathcal{CSHI}$ thus has the *analytic superformula property* [5].

4 Completeness

4.1 Proving Completeness Via Model Graphs

We prove completeness of our calculus via model graphs following [11,5,10]. A model graph is a tuple $\langle \Delta, \tau, \mathcal{C}, \mathcal{E} \rangle$, where: Δ is a finite set; τ is a distinguished element of Δ; $\mathcal{C}$ is a function that maps each element of Δ to a set of concepts; and $\mathcal{E}$ is a function that maps each role name to a binary relation on Δ. A model graph $\langle \Delta, \tau, \mathcal{C}, \mathcal{E} \rangle$ is *saturated* if every $x \in \Delta$ satisfies:

1. if $C \sqcap D \in \mathcal{C}(x)$ then $\{C, D\} \subseteq \mathcal{C}(x)$,
2. if $C \sqcup D \in \mathcal{C}(x)$ then $C \in \mathcal{C}(x)$ or $D \in \mathcal{C}(x)$,

3. if ($\forall R.C \in \mathcal{C}(x)$ or $\forall^\dagger R.C \in \mathcal{C}(x)$) and $\mathcal{E}(R)(x, y)$ holds then $C \in \mathcal{C}(y)$,
4. if $\exists R.C \in \mathcal{C}(x)$ then there exists $y \in \Delta$ with $\mathcal{E}(R)(x, y)$ and $C \in \mathcal{C}(y)$.

A saturated model graph $\langle \Delta, \tau, \mathcal{C}, \mathcal{E}\rangle$ is *consistent* if no $x \in \Delta$ has a $\mathcal{C}(x)$ containing $\bot$ or containing a pair C, $\overline{C}$ for some concept C. Given a model graph $M = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}\rangle$, the *interpretation corresponding to* M is the interpretation $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}}\rangle$ where $A^{\mathcal{I}} = \{x \in \Delta \mid A \in \mathcal{C}(x)\}$ for every atomic concept A and $R^{\mathcal{I}} = \mathcal{E}(R)$ for every role name R.

Lemma 2. *If $\mathcal{I}$ is the interpretation corresponding to a consistent saturated model graph $\langle \Delta, \tau, \mathcal{C}, \mathcal{E}\rangle$, then for every $x \in \Delta$ and $C \in \mathcal{C}(x)$ we have $x \in C^{\mathcal{I}}$.*

Proof. By induction on the structure of C.

Given a $\mathcal{SHI}$ RBox $\mathcal{R}$, a TBox $\mathcal{T}$, and a finite set X of concepts consistent w.r.t. $(\mathcal{R}, \mathcal{T})$, we construct a model of $\mathcal{R}$ and $\mathcal{T}$ that satisfies X by constructing a consistent saturated model graph $\langle \Delta, \tau, \mathcal{C}, \mathcal{E}\rangle$ such that the corresponding interpretation is a model of $\mathcal{R}$, $X \subseteq \mathcal{C}(\tau)$, and $\mathcal{T} \subseteq \mathcal{C}(x)$ for every $x \in \Delta$.

4.2 Saturation

For a finite set $X \supseteq \mathcal{T}$ of concepts that is consistent w.r.t. $(\mathcal{R}, \mathcal{T})$, a set Y of concepts is called a *saturation of* X w.r.t. $(\mathcal{R}, \mathcal{T})$ if Y is a maximal set consistent w.r.t. $(\mathcal{R}, \mathcal{T})$ that is obtainable from X (as a leaf node) by applications of the static rules.

Lemma 3. *Let X be a finite set of concepts consistent w.r.t. $(\mathcal{R}, \mathcal{T})$, and Y a saturation of X w.r.t. $(\mathcal{R}, \mathcal{T})$. Then $X \subseteq Y \subseteq \mathrm{Cl}(\mathcal{R}, \mathcal{T}, X)$ and Y is closed w.r.t. the static rules. Furthermore, there is an effective procedure that constructs such a set Y from X and $(\mathcal{R}, \mathcal{T})$.*

Proof. It is clear that $X \subseteq Y \subseteq \mathrm{Cl}(\mathcal{R}, \mathcal{T}, X)$. Observe that if a static rule is applicable to Y, then one of the corresponding instances of the denominators is consistent w.r.t. $(\mathcal{R}, \mathcal{T})$. Since Y is a saturation of X w.r.t. $(\mathcal{R}, \mathcal{T})$, it is closed w.r.t. the static rules.

We construct a saturation of X w.r.t. $(\mathcal{R}, \mathcal{T})$ as follows: let $Y := X$; while a static rule is applicable to Y and has a corresponding denominator instance Z which is consistent w.r.t. $(\mathcal{R}, \mathcal{T})$ and strictly contains Y, set $Y := Z$. At each iteration, $Y \subset Z \subseteq \mathrm{Cl}(\mathcal{R}, \mathcal{T}, X)$, so this process always terminates. Clearly, the resulting set Y is a saturation of X w.r.t. $(\mathcal{R}, \mathcal{T})$.

4.3 Constructing Model Graphs

Figure 3 contains an algorithm for constructing a model graph. Algorithm 1 assumes that X is consistent w.r.t. $(\mathcal{R}, \mathcal{T})$ and constructs a model of $\mathcal{R}$ and $\mathcal{T}$ that satisfies X. Algorithm 1 always terminates because each $x \in \Delta$ has a unique finite set $\mathcal{C}(x)$, which is a subset of the finite set $\mathrm{Cl}(\mathcal{R}, \mathcal{T}, X)$, so eventually Step 2(a)ii always finds a proxy.

Algorithm 1

```
Input: a $\mathcal{SHI}$ RBox $\mathcal{R}$, a TBox $\mathcal{T}$, and a finite set $X$ of concepts,
    where $X$ is consistent w.r.t. $(\mathcal{R}, \mathcal{T})$.
Output: a model graph $M = \langle \Delta, \tau, \mathcal{C}, \mathcal{E} \rangle$.

1. For an arbitrary node name $\tau$, let $\Delta := \{\tau\}$, and $\mathcal{E}'(R) := \emptyset$ for every role name $R$.
   Let $\mathcal{C}(\tau)$ be a saturation of $X \cup \mathcal{T}$ w.r.t. $(\mathcal{R}, \mathcal{T})$ and mark $\tau$ as unexpanded.
2. While $\Delta$ contains unexpanded elements, take one, say $x$, and do:
   (a) For every concept $\exists R.C \in \mathcal{C}(x)$:
         i. Let $Y = trans_{\mathcal{R}}(\mathcal{C}(x), R) \cup \{C\} \cup \mathcal{T}$ be the result of applying rule $(\exists R)$
            to $\mathcal{C}(x)$, and let $Z$ be a saturation of $Y$ w.r.t. $(\mathcal{R}, \mathcal{T})$.
        ii. If there is a (proxy) $y \in \Delta$ with $\mathcal{C}(y) = Z$ then add pair $(x, y)$ to $\mathcal{E}'(R)$;
       iii. Else add a new element $y$ with $\mathcal{C}(y) := Z$ to $\Delta$, mark $y$ as unexpanded,
            and add the pair $(x, y)$ to $\mathcal{E}'(R)$.
   (b) Mark $x$ as expanded.
3. Let $\mathcal{E}$ be the least extension of $\mathcal{E}'$ that satisfies the conditions $\mathcal{E}(R^-) = (\mathcal{E}(R))^{-1}$
   for every role name $R$ and the assertions of $\mathcal{R}$.
```

Fig. 3. Constructing a Model Graph (Using Limited Caching)

Lemma 4. *Let $\mathcal{R}$ be a $\mathcal{SHI}$ RBox, $\mathcal{T}$ a TBox, and X a finite set of concepts consistent w.r.t. $(\mathcal{R}, \mathcal{T})$. Let $M = \langle \Delta, \tau, \mathcal{C}, \mathcal{E} \rangle$ be the model graph constructed by Algorithm 1 for $(\mathcal{R}, \mathcal{T}, X)$, and $\mathcal{I}$ be the interpretation corresponding to M. Then $\mathcal{I}$ is a model of $\mathcal{R}$ and $\mathcal{T}$ that satisfies X.*

Proof. We first show that M is a consistent saturated model graph. It is sufficient to show that for every $x, y \in \Delta$, if $(\forall R.C \in \mathcal{C}(x)$ or $\forall^\dagger R.C \in \mathcal{C}(x))$ and $\mathcal{E}(R)(x, y)$ holds then $C \in \mathcal{C}(y)$. We prove the following stronger assertion:

> *For all $x, y \in \Delta$ and all $\mathcal{SHI}$-roles R and S: if $\forall R.C \in \mathcal{C}(x)$ (resp. $\forall^\dagger R.C \in \mathcal{C}(x)$) and $\mathcal{E}(S)(x, y)$ and $S \sqsubseteq R \in \mathcal{R}$ then (i) $C \in \mathcal{C}(y)$, and (ii) $\forall R.C \in \mathcal{C}(y)$ (resp. $\forall^\dagger R.C \in \mathcal{C}(y)$) if $R \circ R \sqsubseteq R \in \mathcal{R}$.*

We prove this assertion by induction on the number of steps needed to derive $\mathcal{E}(S)(x, y)$ during Step 3 of Algorithm 1. Suppose that $\forall R.C \in \mathcal{C}(x)$ (resp. $\forall^\dagger R.C \in \mathcal{C}(x)$), $S \sqsubseteq R \in \mathcal{R}$, and $\mathcal{E}(S)(x, y)$ holds. There are the following cases to consider:

1. The assertion trivially holds for the base case when $\mathcal{E}'(S)(x, y)$ holds. Thus we created y as an S-successor for x in order to fulfil some eventuality $\exists S.D \in \mathcal{C}(x)$. But when constructing y, the rule $(\exists R)$ puts $C \in \mathcal{C}(y)$ since $S \sqsubseteq R$ and $\forall R.C \in \mathcal{C}(x)$ (resp. $\forall^\dagger R.C \in \mathcal{C}(x)$). Similarly, it puts $\forall R.C \in \mathcal{C}(y)$ (resp. $\forall^\dagger R.C \in \mathcal{C}(y)$) if $R \circ R \sqsubseteq R \in \mathcal{R}$.
2. Case $\mathcal{E}(S)(x, y)$ is derived from $\mathcal{E}(S')(x, y)$ and $S' \sqsubseteq S \in \mathcal{R}$. Since $S \sqsubseteq R \in \mathcal{R}$ by assumption, we have that $S' \sqsubseteq R \in \mathcal{R}$ by RBox 4. The derivation of $\mathcal{E}(S')(x, y)$ is shorter by one step, so it falls under the induction hypothesis (by putting S' for S in the statement of the assertion). The desired conclusions follow immediately.

3. Case $\mathcal{E}(S)(x,y)$ is derived from $\mathcal{E}(S)(x,z)$, $\mathcal{E}(S)(z,y)$, and $S \circ S \sqsubseteq S \in \mathcal{R}$. Assume that $\forall R.C \in \mathcal{C}(x)$ (the case $\forall^\dagger R.C \in \mathcal{C}(x)$ is similar). Since $S \sqsubseteq R$ and $\forall R.C \in \mathcal{C}(x)$, the rule (H) would have put $\forall S.C \in \mathcal{C}(x)$. Applying the inductive hypothesis to $\mathcal{E}(S)(x,z)$ (and putting S for R in the statement of the assertion), we obtain that $\forall S.C \in \mathcal{C}(z)$ since $S \circ S \sqsubseteq S \in \mathcal{R}$. Applying the inductive hypothesis to $\mathcal{E}(S)(z,y)$ (and putting S for R in the statement of the assertion), we obtain that $C \in \mathcal{C}(y)$. If $R \circ R \sqsubseteq R \in \mathcal{R}$ then, by the inductive assumption, $\forall R.C \in \mathcal{C}(z)$, and $\forall R.C \in \mathcal{C}(y)$.
4. Case $\mathcal{E}(S)(x,y)$ is derived from $\mathcal{E}(S^-)(y,x)$. There are two subcases:
   (a) Case $\forall R.C \in \mathcal{C}(x)$. By the (H) rule, $\forall S.C \in \mathcal{C}(x)$. If $C \notin \mathcal{C}(y)$ then, by the (cut_B) rule, $\overline{C} \in \mathcal{C}(y)$ and $\forall^\dagger S^-.\exists S.\overline{C} \in \mathcal{C}(y)$. Applying the inductive assumption to $\mathcal{E}(S^-)(y,x)$ (and putting S^- for both S and R in the statement of the assertion) gives us that $\exists S.\overline{C} \in \mathcal{C}(x)$. But $\{\exists S.\overline{C}, \forall S.C\}$ is an inconsistent subset of $\mathcal{C}(x)$, contradicting the consistency of $\mathcal{C}(x)$, hence $C \in \mathcal{C}(y)$. If $R \circ R \sqsubseteq R \in \mathcal{R}$ and $\forall R.C \notin \mathcal{C}(y)$ then, by the (cut_5) rule, $\exists R.\overline{C} \in \mathcal{C}(y)$ and $\forall^\dagger R^-.\exists R.\overline{C} \in \mathcal{C}(y)$, and by the inductive assumption, together with $\mathcal{E}(S^-)(y,x)$ and $S^- \sqsubseteq R^- \in \mathcal{R}$ it implies that $\exists R.\overline{C} \in \mathcal{C}(x)$, which contradicts $\forall R.C \in \mathcal{C}(x)$.

   (b) Case $\forall^\dagger R.C \in \mathcal{C}(x)$. There are two subcases since we can create such a $\forall^\dagger R$-concept using only the rules (cut_B) and (cut_5):

   (cut_B): Then $C = \exists R^-.\overline{D}$ with $\overline{D} \in \mathcal{C}(x)$, and $\forall R^-.D \in \widetilde{Sf}_{\mathcal{R}}(X \cup \mathcal{T})$. Suppose that $C \notin \mathcal{C}(y)$, i.e. $\exists R^-.\overline{D} \notin \mathcal{C}(y)$. Then, by the rule $(cut_\forall)$, $\forall R^-.D \in \mathcal{C}(y)$. By RBox 2, we have $S^- \sqsubseteq R^- \in \mathcal{R}$ from $S \sqsubseteq R \in \mathcal{R}$. Since $\forall R^-.D \in \mathcal{C}(y)$ and $S^- \sqsubseteq R^- \in \mathcal{R}$ and $\mathcal{E}(S^-)(y,x)$, by the inductive assumption, $D \in \mathcal{C}(x)$, which contradicts $\overline{D} \in \mathcal{C}(x)$. Therefore $C \in \mathcal{C}(y)$.

   Suppose that $R \circ R \sqsubseteq R \in \mathcal{R}$ and $\forall^\dagger R.C \notin \mathcal{C}(y)$, i.e. $\forall^\dagger R.\exists R^-.\overline{D} \notin \mathcal{C}(y)$. By RBox 3, we have $R^- \circ R^- \sqsubseteq R^- \in \mathcal{R}$ too. Then, by the (cut_5) rule, $\forall R^-.D \in \mathcal{C}(y)$. Analogously as for the previous paragraph, it follows that $D \in \mathcal{C}(x)$, which contradicts $\overline{D} \in \mathcal{C}(x)$. Therefore, $R \circ R \sqsubseteq R \in \mathcal{R}$ implies $\forall^\dagger R.C \in \mathcal{C}(y)$.

   (cut_5): Then $C = \exists R^-.\overline{D}$ with $\exists R^-.\overline{D} \in \mathcal{C}(x)$, $\forall R^-.D \in \widetilde{Sf}_{\mathcal{R}}(X \cup \mathcal{T})$ and $R^- \circ R^- \sqsubseteq R^- \in \mathcal{R}$.

   Suppose $C \notin \mathcal{C}(y)$, i.e. $\exists R^-.\overline{D} \notin \mathcal{C}(y)$. Then $\forall R^-.D \in \mathcal{C}(y)$ by the rule $(cut_\forall)$. By the inductive hypothesis, it follows that $\forall R^-.D \in \mathcal{C}(x)$, which contradicts $\exists R^-.\overline{D} \in \mathcal{C}(x)$. Therefore $C \in \mathcal{C}(y)$.

   Suppose $\forall^\dagger R.C \notin \mathcal{C}(y)$, i.e. $\forall^\dagger R.\exists R^-.\overline{D} \notin \mathcal{C}(y)$. Then $\forall R^-.D \in \mathcal{C}(y)$ by the (cut_5) rule. By the inductive assumption, it follows that $\forall R^-.D \in \mathcal{C}(x)$, which contradicts $\exists R^-.\overline{D} \in \mathcal{C}(x)$. Therefore $\forall^\dagger R.C \in \mathcal{C}(y)$.

By the construction of M, the corresponding interpretation $\mathcal{I}$ is a model of $\mathcal{R}$, and we have that $X \subseteq \mathcal{C}(\tau)$ and $\mathcal{T} \subseteq \mathcal{C}(x)$ for every $x \in \Delta$. Hence, by Lemma 2, $\mathcal{I}$ is a model of $\mathcal{R}$ and $\mathcal{T}$ that satisfies X.

The following theorem immediately follows from Lemmas 1 and 4.

Theorem 1. *The calculus $\mathcal{CSHI}$ is sound and complete.*

5 A Simple EXPTIME Decision Procedure for $\mathcal{SHI}$

The naive decision procedure for $\mathcal{SHI}$ that explores tableaux in the usual way has co-2NEXPTIME complexity because each branch in a tableau may have an exponential length, meaning that the whole tableau (tree) may have a double-exponential number of nodes. Algorithm 1 has the same complexity because of the computation of saturations (using naive tableaux).

By simulating the creation of saturations and checking whether the resulting model graph is consistent, it is easy to alter Algorithm 1 so that it explicitly checks, rather than assumes, that X is consistent w.r.t. $(\mathcal{R}, \mathcal{T})$. Simulating the creation of (a *candidate* for) a saturation is done nondeterministically in linear time. Since $\mathrm{Cl}(\mathcal{R}, \mathcal{T}, X)$ has size $O(n^3)$, the constructed model graph has size $2^{O(n^3)}$. So the resulting algorithm for checking whether X is consistent w.r.t. $(\mathcal{R}, \mathcal{T})$, where $\mathcal{R}$ is a $\mathcal{SHI}$ RBox and $\mathcal{T}$ is a TBox $\mathcal{T}$, runs in NEXPTIME.

In Figure 4 we present an EXPTIME decision procedure for $\mathcal{SHI}$ which directly uses the tableau rules of $\mathcal{CSHI}$ to create an and-or graph as follows.

A node in the constructed and-or graph is a record with three attributes:

content: the set of concepts carried by the node
status: {`unexpanded`, `expanded`, `sat`, `unsat`}
kind: {`and-node`, `or-node`}

To check whether a given finite set X is satisfiable w.r.t. $(\mathcal{R}, \mathcal{T})$, where $\mathcal{R}$ is a $\mathcal{SHI}$ RBox and $\mathcal{T}$ is a TBox, the content of the initial node τ with status `unexpanded` is $X \cup \mathcal{T}$. The main while-loop continues processing nodes until the status of τ is determined to be in {`sat`, `unsat`}, or until every node is expanded, whichever happens first.

Inside the main loop, Steps (2b) to (2f) try to apply one and only one of the tableau rules in the order $(\bot)$, $(\sqcap)$, (H), $(H^\dagger)$, $(\sqcup)$, $(cut_\forall)$, (cut_B), (cut_5), $(\exists R)$ to the current node v. The set $\mathcal{D}$ contains the contents of the resulting denominators of v. If the applied tableau rule is $(\sqcap)$ or (H) or $(H^\dagger)$ then v has one denominator in $\mathcal{D}$; if the applied rule is $(\sqcup)$ or $(cut_\forall)$ or (cut_B) or (cut_5) then v has two denominators in $\mathcal{D}$; otherwise, each concept $\exists R.C \in v.content$ contributes one appropriate denominator to $\mathcal{D}$. At Step (2g), for every denominator in $\mathcal{D}$, we create the required successor in the graph G only if it does not yet exist in the graph: this step merely mimics Algorithm 1 and therefore uses global caching.

In Algorithm 2, a node that contains both C and $\overline{C}$ for some concept C becomes an end-node with status `unsat` (i.e. unsatisfiable w.r.t. $(\mathcal{R}, \mathcal{T})$). A node to which no tableau rule is applicable becomes an end-node with status `sat` (i.e. satisfiable w.r.t. $(\mathcal{R}, \mathcal{T})$).

On the other hand, an application of $(\sqcup)$ or $(cut_\forall)$ or (cut_B) or (cut_5) to a node v causes v to be an *or-node*, while an application of $(\sqcap)$ or (H) or $(H^\dagger)$ or $(\exists R)$ to a node v causes v to be an *and-node*. Steps (2h) and (2i) try to compute the status of such a non-end-node v using the kind (or-node/and-node) of v and the status of the successors of v, treating `unsat` as irrevocably `false` and `sat` as irrevocably `true`.

Algorithm 2

Input: a $\mathcal{SHI}$ RBox $\mathcal{R}$, a TBox $\mathcal{T}$, and a finite set X of concepts

Output: an and-or graph $G = \langle V, E\rangle$ with $\tau \in V$ as the initial node such that $\tau.status = \texttt{sat}$ iff X is satisfiable w.r.t. $(\mathcal{R}, \mathcal{T})$

```
1. create a new node $\tau$ with $\tau.content := X \cup \mathcal{T}$ and $\tau.status := \texttt{unexpanded}$;
   let $V := \{\tau\}$ and $E := \emptyset$;
2. while $\tau.status \notin \{\texttt{sat}, \texttt{unsat}\}$ and we can choose an unexpanded node $v \in V$ do:
   (a) $\mathcal{D} := \emptyset$;
   (b) if no $\mathcal{CSHI}$-tableau rule is applicable to $v.content$ then $v.status := \texttt{sat}$
   (c) else if $(\bot)$ is applicable to $v.content$ then $v.status := \texttt{unsat}$
   (d) else if $(\sqcap)$ or $(H)$ or $(H^\dagger)$ is applicable to $v.content$ giving denominator $Y$ then
           $v.kind := \texttt{and-node}$, $\mathcal{D} := \{Y\}$
   (e) else if $(\sqcup)$ or $(cut_\forall)$ or $(cut_B)$ or $(cut_5)$ is applicable to $v.content$ giving denominators $Y_1$ and $Y_2$ (both different from $v.content$) then
           $v.kind := \texttt{or-node}$, $\mathcal{D} := \{Y_1, Y_2\}$
   (f) else
       i. $v.kind := \texttt{and-node}$,
       ii. for every $\exists R.C \in v.content$, apply $(\exists R)$ to $v.content$ giving denominator
           $trans_{\mathcal{R}}(v.content, R) \cup \{C\} \cup \mathcal{T}$ and add this denominator to $\mathcal{D}$;
   (g) for every denominator $Y \in \mathcal{D}$ do
       i. if some (proxy) $w \in V$ has $w.content = Y$ then add edge $(v, w)$ to $E$
       ii. else let $w$ be a new node, set $w.content := Y$, $w.status := \texttt{unexpanded}$,
           add $w$ to $V$, and add edge $(v, w)$ to $E$;
   (h) if ($v.kind = \texttt{or-node}$ and one of the successors of $v$ has status $\texttt{sat}$)
       or ($v.kind = \texttt{and-node}$ and all the successors of $v$ have status $\texttt{sat}$) then
           $v.status := \texttt{sat}$, $propagate(G, v)$
   (i) else if ($v.kind = \texttt{and-node}$ and one of the successors of $v$ has status $\texttt{unsat}$)
       or ($v.kind = \texttt{or-node}$ and all the successors of $v$ have status $\texttt{unsat}$) then
           $v.status := \texttt{unsat}$, $propagate(G, v)$
   (j) else $v.status := \texttt{expanded}$;
3. if $\tau.status \notin \{\texttt{sat}, \texttt{unsat}\}$ then
       for every node $v \in V$ with $v.status \neq \texttt{unsat}$, set $v.status := \texttt{sat}$;
```

Fig. 4. A Simple EXPTIME Decision Procedure for $\mathcal{SHI}$

If these steps cannot determine the status of v as `sat` or `unsat`, then its status is set to `expanded`. But if these steps do determine the status of a node v to be `sat` or `unsat`, this information is itself propagated to the predecessors of v in the and-or graph G via the routine *propagate*(G, v), explained shortly.

The main loop ends when the status of the initial node τ becomes `sat` or `unsat` or all nodes of the graph have been expanded. In the latter case, all nodes with status $\neq$ `unsat` are given status `sat` (effectively giving the status *open* to tableau branches which loop).

The procedure *propagate* used in the above algorithm is specified in Figure 5. As parameters, it accepts an and-or graph G and a node v with (irrevocable)

```
Procedure $propagate(G, v)$
Parameters: an and-or graph $G = \langle V, E\rangle$ and $v \in V$ with $v.status \in \{\texttt{sat}, \texttt{unsat}\}$
Returns: a modified and-or graph $G = \langle V, E\rangle$

1. $queue := \{v\}$;
2. while $queue$ is not empty do
3. (a) extract $x$ from $queue$;
   (b) for every $u \in V$ with $(u, x) \in E$ and $u.status = \texttt{expanded}$ do
         i. if ($u.kind = \texttt{or-node}$ and one of the successors of $u$ has status sat)
            or ($u.kind = \texttt{and-node}$ and all the successors of $u$ have status sat) then
               $u.status := \texttt{sat}$,  $queue := queue \cup \{u\}$
        ii. else if ($u.kind = \texttt{and-node}$ and one of the successors of $u$ has status unsat)
            or ($u.kind = \texttt{or-node}$ and all the successors of $u$ have status unsat) then
               $u.status := \texttt{unsat}$,  $queue := queue \cup \{u\}$;
```

Fig. 5. Propagating Satisfiability and Unsatisfiability Through an And-Or Graph

status `sat` or `unsat`. The purpose is to propagate the status of v through the and-or graph and alter G to reflect the new information.

Initially, the queue of nodes to be processed contains only v. While the queue is not empty: a node x is extracted; the status of x is propagated to each predecessor u of x in an appropriate way; and if the status of u becomes (irrevocably) `sat` or `unsat` then u is inserted into the queue for further propagation.

This construction thus uses both caching and propagation techniques.

Proposition 1. *Algorithm 2 runs in EXPTIME.*

Proof. Let $G = \langle V, E\rangle$ be the graph constructed by Algorithm 2 for $(\mathcal{R}, \mathcal{T}, X)$ and n be the size of the input, i.e. the sum of the lengths of the concepts of $X \cup \mathcal{T}$ and the lengths of the assertions of $\mathcal{R}$.

For every $v \in V$, we have that $v.content \subseteq \mathrm{Cl}(\mathcal{R}, \mathcal{T}, X)$, hence the size of $v.content$ is $O(n^3)$. For every $v, v' \in V$, if $v \neq v'$ then $v.content \neq v'.content$. Hence V contains $2^{O(n^3)}$ nodes. Every $v \in V$ is expanded (by Steps (2a)–(2j)) only once and such a task takes $2^{O(n^3)}$ time units without counting the execution time of the procedure *propagate*. When $v.status$ becomes `sat` or `unsat`, the procedure *propagate* executes $2^{O(n^3)}$ basic steps directly involved with v. Hence the total time of the executions of *propagate* is of rank $2^{O(n^3)}$. The time complexity of Algorithm 2 is therefore of rank $2^{O(n^3)}$.

Lemma 5. *It is an invariant of Algorithm 2 that for every $v \in V$:*

1. *if $v.status =$ `unsat` then*
   - *$v.content$ contains both C and $\overline{C}$ for some concept C,*
   - *or $v.kind =$ `and-node` and there exists $(v, w) \in E$ such that $w \neq v$ and $w.status =$ `unsat`,*
   - *or $v.kind =$ `or-node` and for every $(v, w) \in E$, $w.status =$ `unsat`;*

2. if $v.status = \texttt{sat}$ then
   - *no $\mathcal{CSHI}$-tableau rule is applicable to $v.content$,*
   - *or $v.kind = \texttt{or-node}$ and there exists $(v, w) \in E$ with $w.status = \texttt{sat}$,*
   - *or $v.kind = \texttt{and-node}$ and for every $(v, w) \in E$, $w.status = \texttt{sat}$.*

(Since a static rule is applied to X only when X is not closed w.r.t. the rule, if $v.kind = \texttt{or-node}$ and $(v, w) \in E$ then $w \neq v$ since $w.content \neq v.content$.)

Proof. Lemma 5(1) clearly holds since these are the only three ways for a node to get status `unsat`. For Lemma 5(2) there is the possibility that the node gets status `sat` via Step 3 of Algorithm 2.

For a contradiction, assume that $v.status$ becomes `sat` because of Step 3 of Algorithm 2 and that all three clauses of the "then" part of Lemma 5(2) fail:

1. First, the rule assumed to be applicable to $v.content$ cannot be the $(\bot)$-rule as this would have put $v.status = \texttt{unsat}$, contradicting our assumption that $v.status = \texttt{sat}$. Thus the rule must be one of the remaining rules, meaning that $v.kind = \texttt{or-node}$ or $v.kind = \texttt{and-node}$ after this rule application.
2. Second, if $v.kind = \texttt{or-node}$ then v must have two successors created by one of the rules $(\sqcup)$, $(cut_\forall)$, (cut_B), (cut_5), since this is the only way for a node to have $v.kind = \texttt{or-node}$. If neither successor has status `sat` then they must both have status `unsat`. But Algorithm 2 and procedure *propagate* always ensure that `unsat` is propagated whenever it is found. As soon as the `unsat` status of the latter of these two children is found, the ensuing call to *propagate* would have ensured that $v.status = \texttt{unsat}$, contradicting our assumption that $v.status = \texttt{sat}$.
3. Third, if $v.kind = \texttt{and-node}$ then v has at least one successor w (say) with $(v, w) \in E$. If $w.status \neq \texttt{sat}$, then we must have $w.status = \texttt{unsat}$. But again, as soon as w gets status `unsat`, procedure *propagate* would ensure that $v.status = \texttt{unsat}$ too, contradicting our assumption that $v.status = \texttt{sat}$.

Lemma 6. *Let $G = \langle V, E\rangle$ be the graph constructed by Algorithm 2 for $(\mathcal{R}, \mathcal{T}, X)$. For every $v \in V$, if $v.status = \texttt{unsat}$ then $v.content$ is inconsistent w.r.t. $(\mathcal{R}, \mathcal{T})$.*

Proof. Using Lemma 5, we can construct a closed tableau w.r.t. $(\mathcal{R}, \mathcal{T})$ for $v.content$ by induction on the way v depends on its successors and by copying nodes so that the resulting structure is a (tree) tableau rather than a graph.

Let $G = \langle V, E\rangle$ be the graph constructed by Algorithm 2 for $(\mathcal{R}, \mathcal{T}, X)$. For $v \in V$ with $v.status = \texttt{sat}$, we say that $v_0 = v, v_1, \ldots, v_k$ with $k \geq 0$ is a *saturation path of v in G* if for each $1 \leq i \leq k$, we have $v_i.status = \texttt{sat}$, the edge $E(v_{i-1}, v_i)$ was created by an application of a static rule, and $v_k.content$ is closed w.r.t. the static rules. Observe that if $v_0, \ldots, v_k$ is a saturation path of v_0 in G then $v_0.content \subseteq \ldots \subseteq v_k.content$.

By Lemma 5, if $v.status = \texttt{sat}$ then there exists a saturation path of v in G.

Lemma 7. *Let $G = \langle V, E\rangle$ be the graph constructed by Algorithm 2 for $(\mathcal{R}, \mathcal{T}, X)$. If $\tau.status = \texttt{sat}$ then every tableau for X w.r.t. $(\mathcal{R}, \mathcal{T})$ is open.*

Proof. Let T be an arbitrary tableau for X w.r.t. $(\mathcal{R}, \mathcal{T})$. We maintain a *current node cn* of T that will follow edges of T to pin-point an open branch of T. Initially we set *cn* to be the root of T. We also keep a (finite) saturation path σ of the form $\sigma_0, \ldots, \sigma_k$ for some $\sigma_0 \in V$ and call σ the *current saturation path in G*. At the beginning, set $\sigma_0 := \tau$ and let σ be a saturation path for σ_0 in G: such a saturation path exists since $\tau.status = \mathtt{sat}$. We maintain the invariant $cn.content \subseteq \sigma_k.content$, where $cn.content$ is the set carried by *cn*.

Remark 2. By the definition of saturation path, $\sigma_k.status = \mathtt{sat}$. The invariant thus implies that the rule $(\bot)$ is not applicable to *cn*.

Clearly, the invariant holds at the beginning since $\sigma_0 = \tau$ and $\tau.content = cn.content$ and $\sigma_0.content \subseteq \sigma_k.content$. Depending upon the rule applied to *cn* in the tableau T, we maintain the invariant by changing the value of the current node *cn* of T and possibly also the current saturation path σ in G:

1. Case the tableau rule applied to *cn* is a static rule. Since $cn.content \subseteq \sigma_k.content$ and $\sigma_k.content$ is closed w.r.t. the static rules, *cn* has a successor u in T with $u.content \subseteq \sigma_k.content$. By setting $cn := u$, the invariant is maintained without changing σ.
2. Case the tableau rule applied to *cn* is the transitional rule $(\exists R)$ with principal concept $\exists R.D$, and the successor is $u \in T$.

   By the invariant, $\exists R.D \in \sigma_k.content$. So there must be a node $w \in V$ such that the edge $E(\sigma_k, w)$ was created by the application of $(\exists R)$ to $\sigma_k.content$ with $\exists R.D$ as the principal concept. Thus, $u.content \subseteq w.content$ by the form of the $(\exists R)$-rule. Moreover, σ_k is an and-node with $\sigma_k.status = \mathtt{sat}$, hence $w.status \neq \mathtt{unsat}$, meaning that $w.status = \mathtt{sat}$. Setting $cn := u$ and setting σ to be a saturation path of w in G maintains the invariant.

By Remark 2, the branch formed by the instances of *cn* is an open branch of T.

Theorem 2. *Let $\mathcal{R}$ be a $\mathcal{SHI}$ RBox, $\mathcal{T}$ a TBox, and X a finite set of concepts. Let $G = \langle V, E \rangle$ be the graph constructed by Algorithm 2 for $(\mathcal{R}, \mathcal{T}, X)$, with $\tau \in V$ as the initial node. Then X is satisfiable w.r.t. $(\mathcal{R}, \mathcal{T})$ iff $\tau.status = \mathtt{sat}$.*

Proof. By Lemmas 6 and 7, X is consistent w.r.t. $(\mathcal{R}, \mathcal{T})$ iff $\tau.status = \mathtt{sat}$ since $\tau.content = X \cup \mathcal{T}$. Since the calculus $\mathcal{CSHI}$ is sound and complete, it follows that X is satisfiable w.r.t. $(\mathcal{R}, \mathcal{T})$ iff $\tau.status = \mathtt{sat}$.

Corollary 1. *Algorithm 2 is an EXPTIME decision procedure for $\mathcal{SHI}$.*

Proof. The EXPTIME complexity is established by Proposition 1.

6 Further Work and Conclusions

To the best of our knowledge, we have given the first EXPTIME tableau-based decision procedure for $\mathcal{SHI}$. The two essential features which allow our decision procedure to have EXPTIME complexity are the analytic “future guessing” cut rules in combination with global caching.

Cut rules are usually considered expensive in tableau calculi because of their exponential nature and because their “blind guessing” produces nondeterminism. Although the “blind guessing” aspect is still prevalent in our cut rules, global caching means that an application of a cut rule in our and-or graph creates only an or-node, deterministically in polynomial time in the size of the graph, but does *not* require us to make two copies of the current and-or graph. Nor does it require us to later “determinise” the effects of “look behind with cut” as queried in the introduction. Indeed, by building a “use-check” into our procedure, and exploring the left branch of a cut rule first, we can avoid exploration of the right branch if the cut formula is not used in closing the left branch. Whether our method can be turned into practical EXPTIME decision procedures requires further investigation, but it obeys the basic principle that practical algorithms should be easy to implement and optimise. Besides, most known optimisation techniques for DL decision procedures are applicable since they are already incorporated into our decision procedure for $\mathcal{ALC}$ in [6], and it is easy to show that they transfer to our decision procedure for $\mathcal{SHI}$.

Soundness of global caching for $\mathcal{SHI}$ was not previously proved and was really an open problem. The idea of using analytic cut rules to reduce the complexity from NEXPTIME to EXPTIME was introduced in [2], but the “future guessing” form of our analytic cut rules is important and fundamentally different from the “look behind analytic cut” of [2].

Observe that the transformation of our $\mathcal{CSHI}$ calculus into an EXPTIME decision procedure for $\mathcal{SHI}$ is highly independent of the details of the calculus. The proof of the correctness of the transformation is also highly modular w.r.t. the calculus. We intend to formulate both at a more abstract level to apply our method for global caching to other modal and description logics.

References

1. Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Studia Logica 69, 5–40 (2001)
2. De Giacomo, G., Massacci, F.: Combining deduction and model checking into tableaux and algorithms for Converse-PDL. Information and Computation, pp. 117–137 pp. 87–138 (2000)
3. Ding, Y., Haarslev, V.: Tableau caching for description logics with inverse and transitive roles. In: Proc. DL-2006: International Workshop on Description Logics, pp. 143–149 (2006)
4. Donini, F., Massacci, F.: EXPTIME tableaux for ALC. Artificial Intelligence 124, 87–138 (2000)
5. Goré, R.: Tableau methods for modal and temporal logics. In: Agostino, D. (ed.) Handbook of Tableau Methods, pp. 297–396. Kluwer, Dordrecht (1999)
6. Goré, R., Nguyen, L.A.: Optimised EXPTIME tableaux for ALC using sound global caching, propagation and cutoffs. Manuscript (2007)

7. Heuerding, A., Seyfried, M., Zimmermann, H.: Efficient loop-check for backward proof search in some non-classical logics. In: Miglioli, P., Moscato, U., Ornaghi, M., Mundici, D. (eds.) TABLEAUX 1996. LNCS, vol. 1071, pp. 210–225. Springer, Heidelberg (1996)
8. Horrocks, I., Patel-Schneider, P.F.: Optimizing description logic subsumption. Journal of Logic and Computation 9(3), 267–293 (1999)
9. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. Log. Comput. 9(3), 385–410 (1999)
10. Nguyen, L.A.: Analytic tableau systems and interpolation for the modal logics KB, KDB, K5, KD5. Studia Logica 69(1), 41–57 (2001)
11. Rautenberg, W.: Modal tableau calculi and interpolation. JPL 12, 403–423 (1983)

Tree-Sequent Methods for Subintuitionistic Predicate Logics

Ryo Ishigaki[1] and Kentaro Kikuchi[2]

[1] Department of Mathematical and Computing Sciences,
Tokyo Institute of Technology, Japan
[2] Research Institute of Electrical Communication,
Tohoku University, Japan

Abstract. Subintuitionistic logics are a class of logics defined by using Kripke models with more general conditions than those for intuitionistic logic. In this paper we study predicate logics of this kind by the method of tree-sequent calculus (a special form of Labelled Deductive System). After proving the completeness with respect to some classes of Kripke models, we introduce Hilbert-style axiom systems and prove their completeness through a translation from the tree-sequent calculi. This gives a solution to the problem posed by Restall.

1 Introduction

Subintuitionistic logics are a class of logics defined by using Kripke models with more general conditions than those for intuitionistic logic. They are interpreted in Kripke models similarly to intuitionistic logic, except that some of the conditions (reflexivity, transitivity and persistence) on models are not imposed. The propositional part of these logics has been studied, for example, in [3,5,6,8,15,18,19]. Some of them are considered as the strict implication fragments of the corresponding modal logics, but others such as those in [18] are not.

In [15], Restall described difficulties in extending subintuitionistic logics to the first-order predicate case. One of the difficulties is that if we take the same clauses in the definition of Kripke models as the standard ones for intuitionistic predicate logic then, for instance, the formula $\forall x A(x) \rightarrow A(a)$ is no longer valid. This is easily seen in Kripke models whose accessibility relation is not reflexive. So Restall suggested that the domain of quantification should be the domain of the world where the quantified formula is interpreted, and that Kripke models under consideration should have constant domains. However, then proving completeness turned out to be much more complicated than the propositional case, and was left as an open problem.

A solution to axiomatizing a predicate extension of a subintuitionistic logic was given in [20] only for the strict implication fragment of the modal logic $\mathsf{S4}$, where Zimmermann proved the completeness of a Hilbert-style system using a Henkin construction. Note that in the case of the strict implication fragment of $\mathsf{S4}$, one can use a restricted form of the deduction theorem, but it is not clear

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 149–164, 2007.

whether the same method works for other predicate extensions of subintuitionistic logics where the deduction theorem is more restricted.

In this paper, we take a different approach to the problem posed by Restall. We systematically study predicate extensions of several subintuitionistic logics by the method of tree-sequent calculus [12], which is a special form of Labelled Deductive System [7]. This style of formulation is useful in axiomatizing various logics defined through Kripke models without regard to intuitionistic or modal languages (see, e.g. [9,10,17]). We prove the completeness of the tree-sequent calculi by constructing a counter model called a saturated tree-sequent. Then we introduce Hilbert-style axiom systems, and define a translation of tree-sequents into formulas to show that each inference rule of the tree-sequent calculi is simulated in the corresponding Hilbert-style systems. This yields the completeness of the Hilbert-style systems with respect to the corresponding classes of Kripke models.

In previous work [10], we presented a tree-sequent calculus for a predicate extension of Visser's propositional logic [18] whose Kripke models have expanding domains. However, no Hilbert-style system for the predicate logic has been obtained. One of the points worth mentioning is that in proving completeness through tree-sequent calculi, models with constant domains are easier to handle than models with expanding domains, while in the usual Henkin construction they are more difficult the other way around.

The organization of the paper is as follows. In Section 2 we define subintuitionistic logics by using Kripke models. In Section 3 we introduce tree-sequent calculi. In Section 4 we prove the completeness of the tree-sequent calculi by a construction of saturated tree-sequents. In Section 5 we introduce Hilbert-style systems and prove the completeness through a translation from the tree-sequent calculi. In Section 6 we conclude with a discussion of related work.

To save space we omit some proofs in Section 5, but a full version with all proofs is available at `http://www.nue.riec.tohoku.ac.jp/user/kentaro/`.

Notation. Our language $\mathcal{L}$ has the following symbols: countably many variables $x_1, x_2, \ldots$; countably many m-ary predicate symbols $p_1^m, p_2^m, \ldots$ for each $m \in \mathbb{N}$; and the logical symbols $\bot$, $\wedge$, $\vee$, $\rightarrow$, $\forall$ and $\exists$. (To simplify the argument, we do not consider constants or function symbols.) The set of formulas is constructed from these in the usual way. Parentheses are often omitted using the convention that $\wedge$ and $\vee$ bind more strongly than $\rightarrow$. We use $A, B, C, \ldots$ for formulas.

The symbol $\top$ is defined as an abbreviation of the formula $\bot \rightarrow \bot$. For a finite set Φ of formulas, $\bigwedge \Phi$ (resp. $\bigvee \Phi$) is defined as the conjunction (resp. disjunction) of all formulas in Φ, if Φ is non-empty, otherwise $\top$ (resp. $\bot$).

2 Semantics of Subintuitionistic Logics

Subintuitionistic logics are defined semantically using Kripke models which are more general than models for intuitionistic logic. The propositional part of these logics has been studied in the literature (e.g. [5,6,15]) where the least subintuitionistic logic is interpreted in the same Kripke models as those for the modal logic K. The predicate extensions of subintuitionistic logics we consider here are based on Kripke models with constant domains, following the suggestion in [15].

Below we introduce several predicate extensions of subintuitionistic logics as sets of formulas that are valid in some classes of Kripke models.

Definition 1 (Model). Let W be a non-empty set, R be a binary relation on W, and D be a non-empty set. For each m-ary predicate symbol p and each $a \in W$, we suppose a relation $p^{\mathcal{I}(a)}$ on D^m. Then the quadruple $\langle W, R, D, \mathcal{I} \rangle$ is called a *model*.

A model $\langle W, R, D, \mathcal{I} \rangle$ is said to be (i) *reflexive* if for any $a \in W$, aRa, (ii) *transitive* if for any $a, b, c \in W$, aRb and bRc imply aRc, and (iii) *persistent* if for any predicate symbol p and any $a, b \in W$, aRb implies $p^{\mathcal{I}(a)} \subseteq p^{\mathcal{I}(b)}$.

Let $M = \langle W, R, D, \mathcal{I} \rangle$ be a model, and $\mathcal{L}[D]$ be the extended language obtained from $\mathcal{L}$ by adding a constant symbol $\underline{d}$ for each $d \in D$. The relation $\models_M$ between $a \in W$ and each closed formula of $\mathcal{L}[D]$ is defined inductively as follows:

$$\begin{array}{lll}
a \models_M p(\underline{d_1}, \ldots, \underline{d_m}) & \text{iff} & (d_1, \ldots, d_m) \in p^{\mathcal{I}(a)}; \\
a \not\models_M \bot; & & \\
a \models_M A \wedge B & \text{iff} & a \models_M A \text{ and } a \models_M B; \\
a \models_M A \vee B & \text{iff} & a \models_M A \text{ or } a \models_M B; \\
a \models_M A \to B & \text{iff} & \text{for all } b \in W \text{ with } aRb,\ b \not\models_M A \text{ or } b \models_M B; \\
a \models_M \forall x A & \text{iff} & a \models_M A[\underline{d}/x] \text{ for all } d \in D; \\
a \models_M \exists x A & \text{iff} & a \models_M A[\underline{d}/x] \text{ for some } d \in D.
\end{array}$$

A formula A is *valid* in a model $M = \langle W, R, D, \mathcal{I} \rangle$, if $a \models_M \forall \overrightarrow{x} A$ for all $a \in W$, where $\overrightarrow{x} = \langle x_1, \ldots, x_n \rangle$ is an enumeration of all free variables in A, and $\forall \overrightarrow{x}$ is an abbreviation for $\forall x_1 \cdots \forall x_n$. ($\forall \overrightarrow{x} A$ is the universal closure of A.)

The predicate logic K^I is defined as the set of all formulas that are valid in every model. Also, the predicate logics KT^I, $\mathsf{K4}^I$, $\mathsf{S4}^I$ and BQL^I are defined as the sets of all formulas that are valid in every reflexive model, transitive model, reflexive and transitive model, and transitive and persistent model, respectively. We use the notation $\mathcal{X}$ to denote any of the logics K^I, KT^I, $\mathsf{K4}^I$, $\mathsf{S4}^I$ and BQL^I.

An example of a formula that is not necessarily valid in the models defined above is $(A \to (B \to C)) \to (B \to (A \to C))$.

The following is an immediate consequence of an induction on the structure of closed formulas.

Lemma 1. *Let $M = \langle W, R, D, \mathcal{I} \rangle$ be a transitive and persistent model. Then, for any closed formula A of $\mathcal{L}[D]$ and any $a, b \in W$, if aRb and $a \models_M A$ then $b \models_M A$.*

Proof. By induction on the structure of A. □

3 Tree-Sequent Calculi

In this section we introduce tree-sequent calculi for subintuitionistic logics as a special form of Labelled Deductive Systems [7], where labels are used for representing a tree-structure and its nodes. In the next sections we prove the soundness and completeness of the tree-sequent calculi with respect to the semantics described in the previous section.

The basic idea of tree-sequent comes from manipulating sequents that reflect the structure of Kripke models directly. For that, each tree-sequent consists of a tree of ordinary sequents.

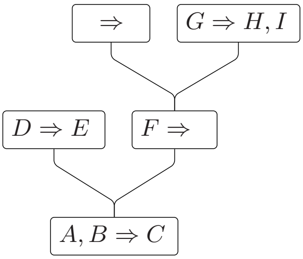
Definition 2 (Tree-sequent). A *label* is a finite sequence of natural numbers $\langle n_1, \ldots, n_m \rangle$. We use letters $\alpha, \beta, \ldots$ for labels. If $\alpha = \langle n_1, \ldots, n_m \rangle$ then $\alpha \cdot n$ denotes the label $\langle n_1, \ldots, n_m, n \rangle$. β is an *immediate successor* (*im-successor*, for short) of α, if $\beta = \alpha \cdot n$ for some natural number n. β is a *successor* of α, if $\beta = ((\cdots((\alpha \cdot n_1) \cdot n_2) \cdots) \cdot n_{m-1}) \cdot n_m$ for some natural numbers $n_1, \ldots, n_m$. A *tree* is a set of labels $\mathcal{T}$ such that $\langle\rangle \in \mathcal{T}$ and for each $\alpha \cdot n \in \mathcal{T}$, $\alpha \in \mathcal{T}$. A *labelled formula* is a pair $\alpha : A$ where α is a label and A is a formula of the language $\mathcal{L}$. A *tree-sequent* is an expression $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ where Γ and Δ are finite sets of labelled formulas, $\mathcal{T}$ is a tree, and each label in Γ, Δ is an element of $\mathcal{T}$.

Thus a tree-sequent $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ represents a finite labelled tree whose structure is induced by $\mathcal{T}$. Each node α of it is associated with a sequent $\Gamma_\alpha \Rightarrow \Delta_\alpha$ where Γ_α (resp. Δ_α) is the set of formulas A such that $\alpha : A \in \Gamma$ (resp. $\alpha : A \in \Delta$).

Example 1. Let $\mathcal{T} = \{\langle\rangle, \langle 1 \rangle, \langle 2 \rangle, \langle 2,1 \rangle, \langle 2,2 \rangle\}$. The tree-sequent

$$\langle\rangle : A, \langle\rangle : B, \langle 1 \rangle : D, \langle 2 \rangle : F, \langle 2,2 \rangle : G \overset{\mathcal{T}}{\Rightarrow} \langle\rangle : C, \langle 1 \rangle : E, \langle 2,2 \rangle : H, \langle 2,2 \rangle : I$$

represents the following tree of sequents:



To make the argument succinct and precise, we use below representation of sequents with labels rather than the tree form as above. A translation of tree-sequents into formulas of the language $\mathcal{L}$ will be defined in Section 5.

Now we introduce tree-sequent calculi for subintuitionistic logics. These systems define inference schemas to manipulate tree-sequents. First we introduce the most basic system $\mathbf{T}\mathsf{K}^I$, which is the tree-sequent calculus for the logic K^I. The axioms (initial tree-sequents) of $\mathbf{T}\mathsf{K}^I$ are of the following forms:

$$\alpha : A, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A \ \text{(Ax)} \qquad \alpha : \bot, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta \ \ (\bot\text{L})$$

The inference rules of $\mathbf{T}\mathsf{K}^I$ are the following:

$$\frac{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\alpha : A, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}\ \text{(Weakening L)} \qquad \frac{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : B}\ \text{(Weakening R)}$$

$$\frac{\alpha : A, \alpha : B, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\alpha : A \wedge B, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}\ (\wedge\text{L}) \qquad \frac{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A \quad \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : B}{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A \wedge B}\ (\wedge\text{R})$$

$$\frac{\alpha : A, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta \quad \alpha : B, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\alpha : A \vee B, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}\ (\vee\text{L}) \qquad \frac{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A, \alpha : B}{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A \vee B}\ (\vee\text{R})$$

$$\frac{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha \cdot n : A \quad \alpha \cdot n : B, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\alpha : A \rightarrow B, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}\ (\rightarrow\text{L})$$

$$\frac{\alpha \cdot n : A, \Gamma \overset{\mathcal{T} \cup \{\alpha \cdot n\}}{\Rightarrow} \Delta, \alpha \cdot n : B}{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A \rightarrow B}\ (\rightarrow\text{R})^{\ddagger}$$

$$\frac{\alpha : A[y/x], \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\alpha : \forall x A, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}\ (\forall\text{L}) \qquad \frac{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A[z/x]}{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : \forall x A}\ (\forall\text{R})_{\text{VC}}$$

$$\frac{\alpha : A[z/x], \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\alpha : \exists x A, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}\ (\exists\text{L})_{\text{VC}} \qquad \frac{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A[y/x]}{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : \exists x A}\ (\exists\text{R})$$

where the subscript VC means the eigenvariable condition: z does not occur in the conclusion. The superscript ‡ means the following condition: $\alpha \cdot n$ does not occur in $\mathcal{T}$.

The above tree-sequent calculus $\mathbf{T}\mathsf{K}^I$ has no cut rule. It also dispenses with contraction rules since a tree-sequent consists of finite sets of labelled formulas.

In Fig. 1, we show the rules $(\rightarrow\text{L})$ and $(\rightarrow\text{R})^{\ddagger}$ with representation in the tree form. In the rule $(\rightarrow\text{L})$, the formula $A \rightarrow B$ is introduced in the antecedent of the parent of the node having the formula A or B in the premisses. Similarly, in the rule $(\rightarrow\text{R})^{\ddagger}$, the formula $A \rightarrow B$ is introduced in the succedent of the parent node. Note that, by the condition of the rule $(\rightarrow\text{R})^{\ddagger}$, the node having the sequent $A \Rightarrow B$ in the premiss is *trimmed* in the conclusion.

Example 2. The tree-sequent $\overset{\{\langle\rangle\}}{\Rightarrow} \langle\rangle : \forall x(A \rightarrow B) \rightarrow (A \rightarrow \forall x B)$, where x is not free in A, is provable in $\mathbf{T}\mathsf{K}^I$ as follows. Let $\mathcal{T} = \{\langle\rangle, \langle 1\rangle, \langle 1, 1\rangle\}$. Then

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle 1,1\rangle : A \overset{\mathcal{T}}{\Rightarrow} \langle 1,1\rangle : B, \langle 1,1\rangle : A \qquad \langle 1,1\rangle : B, \langle 1,1\rangle : A \overset{\mathcal{T}}{\Rightarrow} \langle 1,1\rangle : B}{\langle 1\rangle : A \rightarrow B, \langle 1,1\rangle : A \overset{\mathcal{T}}{\Rightarrow} \langle 1,1\rangle : B}\ (\rightarrow\text{L})}{\langle 1\rangle : \forall x(A \rightarrow B), \langle 1,1\rangle : A \overset{\mathcal{T}}{\Rightarrow} \langle 1,1\rangle : B}\ (\forall\text{L})}{\langle 1\rangle : \forall x(A \rightarrow B), \langle 1,1\rangle : A \overset{\mathcal{T}}{\Rightarrow} \langle 1,1\rangle : \forall x B}\ (\forall\text{R})_{\text{VC}}}{\langle 1\rangle : \forall x(A \rightarrow B) \overset{\{\langle\rangle, \langle 1\rangle\}}{\Rightarrow} \langle 1\rangle : A \rightarrow \forall x B}\ (\rightarrow\text{R})^{\ddagger}}{\overset{\{\langle\rangle\}}{\Rightarrow} \langle\rangle : \forall x(A \rightarrow B) \rightarrow (A \rightarrow \forall x B)}\ (\rightarrow\text{R})^{\ddagger}$$

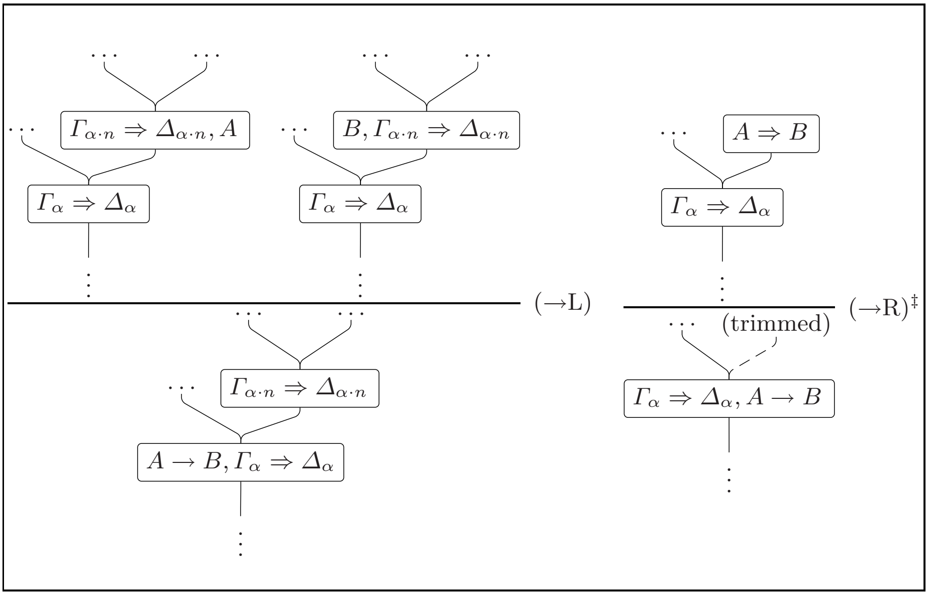

$\Gamma_{\alpha\cdot n} \Rightarrow \Delta_{\alpha\cdot n}, A$
$B, \Gamma_{\alpha\cdot n} \Rightarrow \Delta_{\alpha\cdot n}$
$A \Rightarrow B$
$\Gamma_\alpha \Rightarrow \Delta_\alpha$
$\Gamma_\alpha \Rightarrow \Delta_\alpha$
$\Gamma_\alpha \Rightarrow \Delta_\alpha$
(→L)
(→R)‡
(trimmed)
$\Gamma_{\alpha\cdot n} \Rightarrow \Delta_{\alpha\cdot n}$
$\Gamma_\alpha \Rightarrow \Delta_\alpha, A \to B$
$A \to B, \Gamma_\alpha \Rightarrow \Delta_\alpha$

Fig. 1. (→L) and (→R)‡ in the tree form

The tree-sequent formulation works also for logics with modal language by regarding the formula $\top \to A$ of subintuitionistic logics as $\Box A$ of modal logics. For instance, a tree-sequent calculus for a predicate version of the modal logic K may be defined as a system with the following rules for the modal operator:

$$\frac{\alpha \cdot n : A, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\alpha : \Box A, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}\ (\Box\mathrm{L}) \qquad \frac{\Gamma \overset{\mathcal{T}\cup\{\alpha\cdot n\}}{\Rightarrow} \Delta, \alpha \cdot n : A}{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : \Box A}\ (\Box\mathrm{R})^\ddagger$$

The completeness theorem of the tree-sequent calculus for K is proved similarly to that for $\mathbf{T}\mathsf{K}^I$ in the next section.

Next we introduce tree-sequent calculi for the logics KT^I, $\mathsf{K4}^I$, $\mathsf{S4}^I$ and BQL^I defined in the previous section. The tree-sequent calculus $\mathbf{T}\mathsf{KT}^I$ is obtained from $\mathbf{T}\mathsf{K}^I$ by adding the following rule:

$$\frac{\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A \quad \alpha : B, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\alpha : A \to B, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}\ (\to\mathrm{L}_{\,\mathrm{Ref}})$$

The tree-sequent calculus $\mathbf{T}\mathsf{K4}^I$ is obtained from $\mathbf{T}\mathsf{K}^I$ by adding the following rule:

$$\frac{\alpha \cdot n : A \to B, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\alpha : A \to B, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}\ (\to\mathrm{L}_{\,\mathrm{Tran}})$$

The tree-sequent calculus $\mathbf{T}\mathsf{S4}^I$ is obtained from $\mathbf{T}\mathsf{K}^I$ by adding both of the rules $(\rightarrow\mathrm{L}_{\,\mathrm{Ref}})$ and $(\rightarrow\mathrm{L}_{\,\mathrm{Tran}})$. Finally, the tree-sequent calculus $\mathbf{T}\mathsf{BQL}^I$ is obtained from $\mathbf{T}\mathsf{K}^I$ by adding the following rule:

$$\frac{\alpha \cdot n : A, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}{\alpha : A, \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta}\ (\mathrm{Drop})$$

In the sequel, we use the notation $\mathbf{T}\mathcal{X}$ to denote any of the tree-sequent calculi $\mathbf{T}\mathsf{K}^I$, $\mathbf{T}\mathsf{KT}^I$, $\mathbf{T}\mathsf{K4}^I$, $\mathbf{T}\mathsf{S4}^I$ and $\mathbf{T}\mathsf{BQL}^I$.

4 Completeness of Tree-Sequent Calculi

In this section we show that the tree-sequent calculi introduced in the previous section are sufficient to prove all formulas that are valid in the respective classes of Kripke models. For this purpose we construct a counter model for any formula that is not provable in each tree-sequent calculus. The converse directions, i.e., all provable formulas are valid, are shown through the soundness of Hilbert-style systems in the next section.

In the following, Γ, Δ and $\mathcal{T}$ are possibly infinite in the expression $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ of a tree-sequent. In the case where Γ, Δ and $\mathcal{T}$ are all finite, the tree-sequent $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ is said to be finite. A (possibly infinite) tree-sequent $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ is provable in $\mathbf{T}\mathcal{X}$, if $\Gamma' \overset{\mathcal{T}'}{\Rightarrow} \Delta'$ is provable in $\mathbf{T}\mathcal{X}$ for some finite tree-sequent $\Gamma' \overset{\mathcal{T}'}{\Rightarrow} \Delta'$ such that $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$ and $\mathcal{T}' \subseteq \mathcal{T}$.

Definition 3 ($\mathbf{T}\mathcal{X}$-saturatedness). A tree-sequent $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ is $\mathbf{T}\mathcal{X}$-*saturated*, if it satisfies the following conditions:

- $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ is not provable in $\mathbf{T}\mathcal{X}$;
- (For $\mathbf{T}\mathsf{BQL}^I$ only) If $\alpha : A \in \Gamma$, then $\beta : A \in \Gamma$ for every successor β of α;
- For any $\alpha \in \mathcal{T}$,
  - [$(\wedge\mathrm{L})$] If $\alpha : A \wedge B \in \Gamma$, then $\alpha : A \in \Gamma$ and $\alpha : B \in \Gamma$;
  - [$(\wedge\mathrm{R})$] If $\alpha : A \wedge B \in \Delta$, then $\alpha : A \in \Delta$ or $\alpha : B \in \Delta$;
  - [$(\vee\mathrm{L})$] If $\alpha : A \vee B \in \Gamma$, then $\alpha : A \in \Gamma$ or $\alpha : B \in \Gamma$;
  - [$(\vee\mathrm{R})$] If $\alpha : A \vee B \in \Delta$, then $\alpha : A \in \Delta$ and $\alpha : B \in \Delta$;
  - [$(\rightarrow\mathrm{L})$] If $\alpha : A \rightarrow B \in \Gamma$, then $\alpha \cdot n : A \in \Delta$ or $\alpha \cdot n : B \in \Gamma$ for every im-successor $\alpha \cdot n$ of α;
  - [$(\rightarrow\mathrm{L}_{\,\mathrm{Ref}})$] (For $\mathbf{T}\mathsf{KT}^I/\mathbf{T}\mathsf{S4}^I$ only) If $\alpha : A \rightarrow B \in \Gamma$, then $\alpha : A \in \Delta$ or $\alpha : B \in \Gamma$;
  - [$(\rightarrow\mathrm{L}_{\,\mathrm{Tran}})$] (For $\mathbf{T}\mathsf{K4}^I/\mathbf{T}\mathsf{S4}^I$ only) If $\alpha : A \rightarrow B \in \Gamma$, then $\beta : A \rightarrow B \in \Gamma$ for every successor β of α;
  - [$(\rightarrow\mathrm{R})$] If $\alpha : A \rightarrow B \in \Delta$, then there exists an im-successor $\alpha \cdot n$ of α such that $\alpha \cdot n : A \in \Gamma$ and $\alpha \cdot n : B \in \Delta$;
  - [$(\forall\mathrm{L})$] If $\alpha : \forall x A \in \Gamma$, then $\alpha : A[y/x] \in \Gamma$ for every variable y;
  - [$(\forall\mathrm{R})$] If $\alpha : \forall x A \in \Delta$, then $\alpha : A[z/x] \in \Delta$ for some variable z;
  - [$(\exists\mathrm{L})$] If $\alpha : \exists x A \in \Gamma$, then $\alpha : A[z/x] \in \Gamma$ for some variable z;
  - [$(\exists\mathrm{R})$] If $\alpha : \exists x A \in \Delta$, then $\alpha : A[y/x] \in \Delta$ for every variable y.

The next lemma shows that any unprovable tree-sequent extends to a $\mathbf{T}\mathcal{X}$-saturated tree-sequent, which represents a counter model to the tree-sequent. The process of this extension differs from the usual Henkin-style construction in that it handles the whole counter model rather than separate nodes of a model. The structure of tree-sequents is thus useful in proving completeness of various logics defined through Kripke models.

Lemma 2. *If a finite tree-sequent $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ is not provable in $\mathbf{T}\mathcal{X}$, then there exists a $\mathbf{T}\mathcal{X}$-saturated tree-sequent $\Gamma^+ \overset{\mathcal{T}^+}{\Rightarrow} \Delta^+$ such that $\Gamma \subseteq \Gamma^+$, $\Delta \subseteq \Delta^+$ and $\mathcal{T} \subseteq \mathcal{T}^+$.*

Proof. Suppose that a finite tree-sequent $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ is not provable in $\mathbf{T}\mathcal{X}$. In the following, we construct an infinite sequence of finite tree-sequents $\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1$, $\Gamma^2 \overset{\mathcal{T}^2}{\Rightarrow} \Delta^2, \ldots$ and obtain $\Gamma^+ \overset{\mathcal{T}^+}{\Rightarrow} \Delta^+$ as the union of them.

Let $B_1, B_2, \ldots$ be an enumeration of all formulas of the language $\mathcal{L}$ such that each formula appears infinitely many times. We also enumerate all variables as $x_1, x_2, \ldots$.

Let $\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1 \equiv \Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$. The i-th step, which is the step of extension from $\Gamma^i \overset{\mathcal{T}^i}{\Rightarrow} \Delta^i$ to $\Gamma^{i+1} \overset{\mathcal{T}^{i+1}}{\Rightarrow} \Delta^{i+1}$, consists of operations for the formula B_i. In these operations, unprovability of the tree-sequent is preserved. The operations executed in the i-th step are as follows:

- (For TBQL^I only) For each $\alpha \in \mathcal{T}^i$, if $\alpha : B_i \in \Gamma^i$, then add $\beta : B_i$ to Γ^i for every successor β of α. Unprovability is preserved because of the inference rule (Drop).
- According to the form of B_i, one of the following operations is executed for each label $\alpha \in \mathcal{T}^i$.
  - [B_i is of the form $C \wedge D$]
    If $\alpha : B_i \in \Gamma^i$, then add $\alpha : C$ and $\alpha : D$ to Γ^i. Unprovability is preserved because of the inference rule ($\wedge$L).
    If $\alpha : B_i \in \Delta^i$, then add $\alpha : C$ or $\alpha : D$ to Γ^i, so that unprovability is preserved because of the inference rule ($\wedge$R).
  - [B_i is of the form $C \vee D$]
    Symmetric to the case for $C \wedge D$.
  - [B_i is of the form $C \rightarrow D$]
    (For $\mathsf{TK}^I/\mathsf{TBQL}^I$) If $\alpha : B_i \in \Gamma^i$, then add $\alpha \cdot n : C$ to Δ^i or $\alpha \cdot n : D$ to Γ^i for each im-successor $\alpha \cdot n$ of α, so that unprovability is preserved.
    (For TKT^I) If $\alpha : B_i \in \Gamma^i$, then add $\alpha : C$ to Δ^i or $\alpha : D$ to Γ^i, and add $\alpha \cdot n : C$ to Δ^i or $\alpha \cdot n : D$ to Γ^i for each im-successor $\alpha \cdot n$ of α, so that unprovability is preserved.
    (For $\mathsf{TK4}^I$) If $\alpha : B_i \in \Gamma^i$, then add $\beta : B_i$ to Γ^i for every successor β of α, and add $\alpha \cdot n : C$ to Δ^i or $\alpha \cdot n : D$ to Γ^i for each im-successor $\alpha \cdot n$ of α, so that unprovability is preserved.
    (For $\mathsf{TS4}^I$) If $\alpha : B_i \in \Gamma^i$, then add $\beta : B_i$ to Γ^i for every successor β of α, add $\alpha : C$ to Δ^i or $\alpha : D$ to Γ^i, and add $\alpha \cdot n : C$ to Δ^i or $\alpha \cdot n : D$ to Γ^i for each im-successor $\alpha \cdot n$ of α, so that unprovability is preserved.

(For $\mathbf{T}\mathcal{X}$) If $\alpha : B_i \in \Delta^i$, then add a new im-successor $\alpha \cdot n$ of α to $\mathcal{T}^i$, add $\alpha \cdot n : C$ to Γ^i, and add $\alpha \cdot n : D$ to Δ^i. Unprovability is preserved because of the inference rule $(\rightarrow\mathrm{R})^{\ddagger}$.

- $[B_i$ is of the form $\forall x C]$
  If $\alpha : B_i \in \Gamma^i$, then add $\alpha : C[x_1/x], \ldots, \alpha : C[x_i/x]$ to Γ^i. Unprovability is preserved because of the inference rule $(\forall\mathrm{L})$.
  If $\alpha : B_i \in \Delta^i$, then take a fresh variable z, and add $\alpha : C[z/x]$ to Δ^i. Unprovability is preserved because of the inference rule $(\forall\mathrm{R})_{\mathrm{VC}}$.
- $[B_i$ is of the form $\exists x C]$
  Symmetric to the case for $\forall x C$.

Now let $\Gamma^+ \overset{\mathcal{T}^+}{\Rightarrow} \Delta^+ \equiv \bigcup_{n=1}^{\infty} \Gamma^n \overset{\bigcup_{n=1}^{\infty} \mathcal{T}^n}{\Rightarrow} \bigcup_{n=1}^{\infty} \Delta^n$. It is easy to verify that the tree-sequent $\Gamma^+ \overset{\mathcal{T}^+}{\Rightarrow} \Delta^+$ is $\mathbf{T}\mathcal{X}$-saturated. □

We are now ready to prove the completeness of the tree-sequent calculi.

Theorem 1 (Completeness of $\mathbf{T}\mathcal{X}$). *For any formula A, if $A \in \mathcal{X}$ then the tree-sequent $\overset{\{\langle\rangle\}}{\Rightarrow} \langle\rangle : A$ is provable in $\mathbf{T}\mathcal{X}$.*

Proof. Suppose that $\overset{\{\langle\rangle\}}{\Rightarrow} \langle\rangle : A$ is not provable in $\mathbf{T}\mathcal{X}$. Then by Lemma 2, there exists a $\mathbf{T}\mathcal{X}$-saturated tree-sequent $\Gamma^+ \overset{\mathcal{T}^+}{\Rightarrow} \Delta^+$ such that $\{\langle\rangle : A\} \subseteq \Delta^+$. Now we define a model $M = \langle W, R, D, \mathcal{I}\rangle$ as follows:

- W is $\mathcal{T}^+$;
- (For $\mathbf{T}\mathsf{K}^{\mathit{I}}$) $\alpha R \beta$ iff β is an im-successor of α in $\mathcal{T}^+$;
  (For $\mathbf{T}\mathsf{KT}^{\mathit{I}}$) $\alpha R \beta$ iff $\alpha = \beta$ or β is an im-successor of α in $\mathcal{T}^+$;
  (For $\mathbf{T}\mathsf{K4}^{\mathit{I}}/\mathbf{T}\mathsf{BQL}^{\mathit{I}}$) $\alpha R \beta$ iff β is a successor of α in $\mathcal{T}^+$;
  (For $\mathbf{T}\mathsf{S4}^{\mathit{I}}$) $\alpha R \beta$ iff $\alpha = \beta$ or β is a successor of α in $\mathcal{T}^+$;
- D is the set of all variables;
- $(y_1, \ldots, y_m) \in p^{\mathcal{I}(\alpha)}$ iff $\alpha : p(y_1, \ldots, y_m) \in \Gamma^+$.

It is easy to verify that M satisfies the conditions of respective models.

Now we show by induction on B that for any labelled formula $\alpha : B$,

- if $\alpha : B \in \Gamma^+$ then $\alpha \models_M B[\underline{\overrightarrow{x_B}}/\overrightarrow{x_B}]$, and
- if $\alpha : B \in \Delta^+$ then $\alpha \not\models_M B[\underline{\overrightarrow{x_B}}/\overrightarrow{x_B}]$,

where $\overrightarrow{x_B}$ is an enumeration of all free variables in B. Here we consider only the cases where B is of the form $C \rightarrow D$ and of the form $\forall x C$.

- $[B$ is of the form $C \rightarrow D]$
  (For $\mathbf{T}\mathsf{K}^{\mathit{I}}$) If $\alpha : B \in \Gamma^+$ then by the $\mathbf{T}\mathsf{K}^{\mathit{I}}$-saturatedness, for any im-successor $\alpha \cdot n$ of α, we have $\alpha \cdot n : C \in \Delta^+$ or $\alpha \cdot n : D \in \Gamma^+$. By the induction hypothesis, we have $\alpha \cdot n \not\models_M C[\underline{\overrightarrow{x_C}}/\overrightarrow{x_C}]$ or $\alpha \cdot n \models_M D[\underline{\overrightarrow{x_D}}/\overrightarrow{x_D}]$. Hence, $\alpha \models_M (C \rightarrow D)[\underline{\overrightarrow{x_{C\rightarrow D}}}/\overrightarrow{x_{C\rightarrow D}}]$.
  (For $\mathbf{T}\mathsf{KT}^{\mathit{I}}$) If $\alpha : B \in \Gamma^+$ then by the $\mathbf{T}\mathsf{KT}^{\mathit{I}}$-saturatedness, $\alpha : C \in \Delta^+$ or $\alpha : D \in \Gamma^+$, and for any im-successor $\alpha \cdot n$ of α, we have $\alpha \cdot n : C \in \Delta^+$ or

$\alpha \cdot n : D \in \Gamma^+$. By the induction hypothesis, we have $\alpha \not\models_M C[\underline{\overrightarrow{x_C}}/\overrightarrow{x_C}]$ or $\alpha \models_M D[\underline{\overrightarrow{x_D}}/\overrightarrow{x_D}]$, and $\alpha \cdot n \not\models_M C[\underline{\overrightarrow{x_C}}/\overrightarrow{x_C}]$ or $\alpha \cdot n \models_M D[\underline{\overrightarrow{x_D}}/\overrightarrow{x_D}]$. Hence, $\alpha \models_M (C \to D)[\underline{\overrightarrow{x_{C\to D}}}/\overrightarrow{x_{C\to D}}]$.

(For $\mathbf{T}\mathsf{K4}^I/\mathbf{T}\mathsf{BQL}^I$) If $\alpha : B \in \Gamma^+$ then by the $\mathbf{T}\mathsf{K4}^I/\mathbf{T}\mathsf{BQL}^I$-saturatedness, for any successor β of α, we have $\beta : B \in \Gamma^+$, and $\beta : C \in \Delta^+$ or $\beta : D \in \Gamma^+$. By the induction hypothesis, we have $\beta \not\models_M C[\underline{\overrightarrow{x_C}}/\overrightarrow{x_C}]$ or $\beta \models_M D[\underline{\overrightarrow{x_D}}/\overrightarrow{x_D}]$. Hence, $\alpha \models_M (C \to D)[\underline{\overrightarrow{x_{C\to D}}}/\overrightarrow{x_{C\to D}}]$.

(For $\mathbf{T}\mathsf{S4}^I$) If $\alpha : B \in \Gamma^+$ then by the $\mathbf{T}\mathsf{S4}^I$-saturatedness, $\alpha : C \in \Delta^+$ or $\alpha : D \in \Gamma^+$, and for any successor β of α, we have $\beta : B \in \Gamma^+$, and $\beta : C \in \Delta^+$ or $\beta : D \in \Gamma^+$. By the induction hypothesis, we have $\alpha \not\models_M C[\underline{\overrightarrow{x_C}}/\overrightarrow{x_C}]$ or $\alpha \models_M D[\underline{\overrightarrow{x_D}}/\overrightarrow{x_D}]$, and $\beta \not\models_M C[\underline{\overrightarrow{x_C}}/\overrightarrow{x_C}]$ or $\beta \models_M D[\underline{\overrightarrow{x_D}}/\overrightarrow{x_D}]$. Hence, $\alpha \models_M (C \to D)[\underline{\overrightarrow{x_{C\to D}}}/\overrightarrow{x_{C\to D}}]$.

(For $\mathbf{T}\mathcal{X}$) If $\alpha : B \in \Delta^+$ then by the $\mathbf{T}\mathcal{X}$-saturatedness, there exists an im-successor $\alpha \cdot n$ of α such that $\alpha \cdot n : C \in \Gamma^+$ and $\alpha \cdot n : D \in \Delta^+$. By the induction hypothesis, we have $\alpha \cdot n \models_M C[\underline{\overrightarrow{x_C}}/\overrightarrow{x_C}]$ and $\alpha \cdot n \not\models_M D[\underline{\overrightarrow{x_D}}/\overrightarrow{x_D}]$. Hence, $\alpha \not\models_M (C \to D)[\underline{\overrightarrow{x_{C\to D}}}/\overrightarrow{x_{C\to D}}]$.

- [B is of the form $\forall xC$]

  If $\alpha : B \in \Gamma^+$ then by the $\mathbf{T}\mathcal{X}$-saturatedness, $\alpha : C[y/x] \in \Gamma^+$ for any variable y. By the induction hypothesis, $\alpha \models_M C[y/x][\underline{\overrightarrow{x_{C[y/x]}}}/\overrightarrow{x_{C[y/x]}}]$, which means $\alpha \models_M C[\underline{\overrightarrow{x_C \backslash x}}/\overrightarrow{x_C \backslash x}][\underline{y}/x]$. Hence, we have $\alpha \models_M \forall xC[\underline{\overrightarrow{x_B}}/\overrightarrow{x_B}]$, i.e., $\alpha \models_M (\forall xC)[\underline{\overrightarrow{x_B}}/\overrightarrow{x_B}]$.

  The case where $\alpha : B \in \Delta^+$ is proved similarly.

Since $\langle\rangle : A \in \Delta^+$, we have $\langle\rangle \not\models_M A[\underline{\overrightarrow{x_A}}/\overrightarrow{x_A}]$. Hence, A is not valid in this model M. □

5 Completeness of Hilbert-Style Systems

In this section we introduce Hilbert-style systems for the predicate extensions of subintuitionistic logics, and investigate their relationships with the tree-sequent calculi. Hilbert-style systems for subintuitionistic propositional logics have been studied in [5,6,15]. Here we define a translation of tree-sequents into formulas, and show that each inference rule of the tree-sequent calculi is simulated in the corresponding Hilbert-style systems. This yields the completeness of the Hilbert-style systems with respect to the classes of Kripke models.

First we introduce the system $\mathbf{H}\mathsf{K}^I$, which is a Hilbert-style system for the least subintuitionistic logic K^I. The axioms of $\mathbf{H}\mathsf{K}^I$ are the following:

(A1) $A \to A$,

(A2) $(A \to B) \wedge (B \to C) \to (A \to C)$,

(A3) $A \wedge B \to A$,

(A4) $A \wedge B \to B$,

(A5) $(A \to B) \wedge (A \to C) \to (A \to B \wedge C)$, (A6) $A \to A \vee B$,

(A7) $B \to A \vee B$,

(A8) $(A \to C) \wedge (B \to C) \to (A \vee B \to C)$,

(A9) $A \wedge (B \vee C) \to (A \wedge B) \vee (A \wedge C)$,

(A10) $\bot \to A$,

(A11) $\forall x A \to A[y/x]$,

(A12) $A[y/x] \to \exists x A$,

(A13) $\forall x(A \vee B) \to \forall x A \vee B$ where x is not free in B,

(A14) $\exists x A \wedge B \to \exists x(A \wedge B)$ where x is not free in B,

(A15) $\forall x(B \to A) \to (B \to \forall x A)$ where x is not free in B,

(A16) $\forall x(A \to B) \to (\exists x A \to B)$ where x is not free in B.

The inference rules of $\mathbf{H}\mathsf{K}^I$ are the following:

$$\frac{A \quad A \to B}{B}\ (\mathrm{MP}) \qquad \frac{A}{B \to A}\ (\mathrm{AF}) \qquad \frac{A \quad B}{A \wedge B}\ (\wedge\mathrm{I}) \qquad \frac{A}{\forall x A}\ (\mathrm{GR})$$

Hilbert-style systems for the logics KT^I, $\mathsf{K4}^I$, $\mathsf{S4}^I$ and BQL^I are introduced as follows. The system $\mathbf{H}\mathsf{KT}^I$ is obtained from $\mathbf{H}\mathsf{K}^I$ by adding the following axiom:

(A17) $A \wedge (A \to B) \to B$.

The system $\mathbf{H}\mathsf{K4}^I$ is obtained from $\mathbf{H}\mathsf{K}^I$ by adding the following axiom:

(A18) $(A \to B) \to (C \to (A \to B))$.

The system $\mathbf{H}\mathsf{S4}^I$ is obtained from $\mathbf{H}\mathsf{K}^I$ by adding both of the axioms (A17) and (A18). It is the same in provability as the system of [20], but has more natural axioms on $\vee$ and $\exists$. Finally, the system $\mathbf{H}\mathsf{BQL}^I$ is obtained from $\mathbf{H}\mathsf{K}^I$ by adding the following axiom:

(A19) $A \to (B \to A)$.

In the sequel, we use the notation $\mathbf{H}\mathcal{X}$ to denote any of the Hilbert-style systems $\mathbf{H}\mathsf{K}^I$, $\mathbf{H}\mathsf{KT}^I$, $\mathbf{H}\mathsf{K4}^I$, $\mathbf{H}\mathsf{S4}^I$ and $\mathbf{H}\mathsf{BQL}^I$.

Theorem 2 (Soundness of $\mathbf{H}\mathcal{X}$). *For any formula A, if A is provable in $\mathbf{H}\mathcal{X}$ then $A \in \mathcal{X}$.*

Proof. By induction on the proof of A in $\mathbf{H}\mathcal{X}$. □

In the rest of this section, we prove the completeness of the Hilbert-style systems. The difficulty doing this lies in that the deduction theorem is not available in these systems. So we provide below some derivable rules and formulas instead.

Lemma 3. *The following rules are derivable in* $\mathbf{H}\mathcal{X}$:

$$\frac{A \to B \quad B \to C}{A \to C}\,(\mathrm{Tr}) \qquad \frac{A \to B \quad A \to C}{A \to B \wedge C}\,(\to\wedge\,\mathrm{I})$$

$$\frac{A \to (B \to C) \quad A \to (C \to D)}{A \to (B \to D)}\,(\to\mathrm{Tr})$$

$$\frac{B \to C \quad A \to (C \to D)}{A \to (B \to D)}\,(\mathrm{Tr}\,2) \qquad \frac{A \to (B \to C) \quad C \to D}{A \to (B \to D)}\,(\mathrm{Tr}\,3)$$

$$\frac{A \to B}{(B \to C) \to (A \to C)}\,(\mathrm{Suff}) \qquad \frac{A \to B}{(C \to A) \to (C \to B)}\,(\mathrm{Pref})$$

Note 1. These rules are used in an inductive proof of the equivalent replacement, which justifies such an expression as $\bigwedge \Gamma$ with the associativity and commutativity of $\wedge$ on provability in $\mathbf{H}\mathcal{X}$.

Lemma 4. *The following formulas are provable in* $\mathbf{H}\mathcal{X}$:

1. $(A \to B) \to (A \vee C \to B \vee C)$,
2. $A \wedge (B \vee C) \to B \vee (A \wedge C)$,
3. $(A \vee B) \wedge (A \vee C) \to A \vee (B \wedge C)$.

Lemma 5. *The following rules are derivable in* $\mathbf{H}\mathcal{X}$:

$$\frac{A}{C \to D \vee A}\,(\mathrm{R1}) \qquad \frac{A_1 \to A}{(C \to D \vee A_1) \to (C \to D \vee A)}\,(\mathrm{R2})$$

$$\frac{A_1 \wedge A_2 \to A}{(C \to D \vee A_1) \wedge (C \to D \vee A_2) \to (C \to D \vee A)}\,(\mathrm{R3})$$

Next we define a translation of tree-sequents into formulas. (In the following, tree-sequents are all finite.)

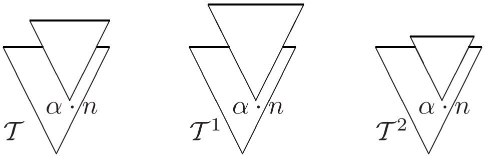
Definition 4 (Formulaic translation). Let $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ be a tree-sequent. For each $\alpha \in \mathcal{T}$, the formula $[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_\alpha$ is defined inductively as follows:

$$[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_\alpha \equiv (\bigwedge \Gamma_\alpha) \to (\bigvee \Delta_\alpha) \vee [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha\cdot n_1} \vee \cdots \vee [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha\cdot n_k}$$

where $\{\alpha\cdot n_1, \ldots, \alpha\cdot n_k\}$ is the set of im-successors of α in $\mathcal{T}$, and Γ_α (resp. Δ_α) is the set of formulas A with $\alpha : A \in \Gamma$ (resp. $\alpha : A \in \Delta$). Then the *formulaic translation* of $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ is defined as the universal closure of $[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\langle\rangle}$.

Now our aim is to show that for any tree-sequent that is provable in one of the tree-sequent calculi, its formulaic translation is provable in the corresponding Hilbert-style system. To facilitate the proof, we give some lemmas.

Lemma 6. *Let* $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$, $\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1$ *and* $\Gamma^2 \overset{\mathcal{T}^2}{\Rightarrow} \Delta^2$ *be tree-sequents that have the same structure except for the parts associated with* $\alpha \cdot n \in \mathcal{T} \cap \mathcal{T}^1 \cap \mathcal{T}^2$ *and all of its successors.*



1. *If* $[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha\cdot n}$ *is provable in* $\mathbf{H}\mathcal{X}$*, then* $[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha}$ *is provable in* $\mathbf{H}\mathcal{X}$*.*
2. *If* $[\![\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1]\!]_{\alpha\cdot n} \to [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha\cdot n}$ *is provable in* $\mathbf{H}\mathcal{X}$*, then* $[\![\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1]\!]_{\alpha} \to [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha}$ *is provable in* $\mathbf{H}\mathcal{X}$*.*
3. *If* $[\![\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1]\!]_{\alpha\cdot n} \wedge [\![\Gamma^2 \overset{\mathcal{T}^2}{\Rightarrow} \Delta^2]\!]_{\alpha\cdot n} \to [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha\cdot n}$ *is provable in* $\mathbf{H}\mathcal{X}$*, then* $[\![\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1]\!]_{\alpha} \wedge [\![\Gamma^2 \overset{\mathcal{T}^2}{\Rightarrow} \Delta^2]\!]_{\alpha} \to [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha}$ *is provable in* $\mathbf{H}\mathcal{X}$*.*

Proof. Direct application of Lemma 5. □

Lemma 7. *Let* $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$*,* $\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1$ *and* $\Gamma^2 \overset{\mathcal{T}^2}{\Rightarrow} \Delta^2$ *be tree-sequents that have the same structure except for the parts associated with* $\alpha \in \mathcal{T} \cap \mathcal{T}^1 \cap \mathcal{T}^2$ *and all of its successors.*

1. *Suppose that* $[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha}$ *is provable in* $\mathbf{H}\mathcal{X}$*. Then,* $[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\langle\rangle}$ *is provable in* $\mathbf{H}\mathcal{X}$*.*
2. *Suppose that* $[\![\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1]\!]_{\alpha} \to [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha}$ *is provable in* $\mathbf{H}\mathcal{X}$*. Then, if* $[\![\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1]\!]_{\langle\rangle}$ *is provable in* $\mathbf{H}\mathcal{X}$ *then* $[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\langle\rangle}$ *is provable in* $\mathbf{H}\mathcal{X}$*.*
3. *Suppose that* $[\![\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1]\!]_{\alpha} \wedge [\![\Gamma^2 \overset{\mathcal{T}^2}{\Rightarrow} \Delta^2]\!]_{\alpha} \to [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\alpha}$ *is provable in* $\mathbf{H}\mathcal{X}$*. Then, if* $[\![\Gamma^1 \overset{\mathcal{T}^1}{\Rightarrow} \Delta^1]\!]_{\langle\rangle}$ *and* $[\![\Gamma^2 \overset{\mathcal{T}^2}{\Rightarrow} \Delta^2]\!]_{\langle\rangle}$ *are provable in* $\mathbf{H}\mathcal{X}$ *then* $[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta]\!]_{\langle\rangle}$ *is provable in* $\mathbf{H}\mathcal{X}$*.*

Proof. Easily seen by applying Lemma 6 as many times as the length of α. □

Lemma 8. *The following formulas are provable in* $\mathbf{H}\mathcal{X}$:

1. $A \wedge C \to D \vee A$,
2. $\bot \wedge C \to D$,
3. $(C \to D) \to (A \wedge C \to D)$,
4. $(C \to D) \to (C \to D \vee B)$,
5. $(C \to D \vee A) \wedge (C \to D \vee B) \to (C \to D \vee (A \wedge B))$,
6. $(A \wedge C \to D) \wedge (B \wedge C \to D) \to ((A \vee B) \wedge C \to D)$,
7. $(C \to D \vee (E \to F \vee A)) \wedge (C \to D \vee (B \wedge E \to F))$ $\to ((A \to B) \wedge C \to D \vee (E \to F))$,
8. $(A[y/x] \wedge C \to D) \to (\forall x A \wedge C \to D)$,
9. $(C \to D \vee A[y/x]) \to (C \to D \vee \exists x A)$.

The following formula is provable in $\mathbf{H}\mathsf{KT}^I/\mathbf{H}\mathsf{S4}^I$:

10. $(C \to D \vee A) \wedge (B \wedge C \to D) \to ((A \to B) \wedge C \to D)$.

The following formula is provable in $\mathbf{H}\mathsf{K4}^I/\mathbf{H}\mathsf{S4}^I$:

11. $(C \to D \vee ((A \to B) \wedge E \to F)) \to ((A \to B) \wedge C \to D \vee (E \to F))$.

The following formula is provable in $\mathbf{H}\mathsf{BQL}^I$:

12. $(C \to D \vee (A \wedge E \to F)) \to (A \wedge C \to D \vee (E \to F))$.

Now we prove a crucial lemma for the completeness of the Hilbert-style systems.

Lemma 9. *For any tree-sequent* $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$*, if* $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ *is provable in* $\mathbf{T}\mathcal{X}$*, then the formulaic translation of* $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ *is provable in* $\mathbf{H}\mathcal{X}$*.*

Proof. By induction on the proof of $\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta$ in $\mathbf{T}\mathcal{X}$. All cases except the rules $(\forall\mathrm{R})_{\mathrm{VC}}$ and $(\exists\mathrm{L})_{\mathrm{VC}}$ immediately follow from Lemmas 7 and 8. Here we consider the case where the last applied rule is $(\forall\mathrm{R})_{\mathrm{VC}}$. Then by the induction hypothesis, the universal closure of $\forall z [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A[z/x]]\!]_{\langle\rangle}$ is provable in $\mathbf{H}\mathcal{X}$, where

$$[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A[z/x]]\!]_{\langle\rangle} \equiv$$
$$(\textstyle\bigwedge \Gamma_{\langle\rangle}) \to (\bigvee \Delta_{\langle\rangle}) \vee [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A[z/x]]\!]_{\langle n_1\rangle} \vee \cdots \vee [\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : A[z/x]]\!]_{\langle n_k\rangle}$$

and $\{\langle n_1\rangle, \ldots, \langle n_k\rangle\}$ is the set of im-successors of $\langle\rangle$ in $\mathcal{T}$. Now, using (A15), (A13) and (Tr 3) repeatedly, we can move $\forall z$ to the front of $A[z/x]$ at the node α. Hence the universal closure of $[\![\Gamma \overset{\mathcal{T}}{\Rightarrow} \Delta, \alpha : \forall z A[z/x]]\!]_{\langle\rangle}$ is provable in $\mathbf{H}\mathcal{X}$.

The case where the last applied rule is $(\exists\mathrm{L})_{\mathrm{VC}}$ is proved similarly using also (A16) and (A14). □

We can now prove the completeness of the Hilbert-style systems as well as the soundness of the tree-sequent calculi.

Theorem 3 (Completeness of $\mathbf{H}\mathcal{X}$, Soundness of $\mathbf{T}\mathcal{X}$). *For any formula* A*, the following are equivalent*:

1. $A \in \mathcal{X}$,
2. A *is provable in* $\mathbf{H}\mathcal{X}$,
3. $\overset{\{\langle\rangle\}}{\Rightarrow} \langle\rangle : A$ *is provable in* $\mathbf{T}\mathcal{X}$.

Proof. (1 ⇒ 3) This follows from Theorem 1.

(3 ⇒ 2) Suppose that the tree-sequent $\overset{\{\langle\rangle\}}{\Rightarrow} \langle\rangle : A$ is provable in $\mathbf{T}\mathcal{X}$. Then by Lemma 9, the universal closure of $[\![\overset{\{\langle\rangle\}}{\Rightarrow} \langle\rangle : A]\!]_{\langle\rangle}$ is provable in $\mathbf{H}\mathcal{X}$, and hence $\top \to A$ is provable in $\mathbf{H}\mathcal{X}$. Therefore, A is provable in $\mathbf{H}\mathcal{X}$.
(2 ⇒ 1) This follows from Theorem 2. □

6 Conclusion

We have studied predicate extensions of subintuitionistic logics by the method of tree-sequent calculus. First we introduced tree-sequent calculi and proved their completeness with respect to the semantics of subintuitionistic logics. Then we defined a translation of tree-sequents into formulas of the intuitionistic language and proved the completeness of new Hilbert-style systems using the translation.

It seems difficult to axiomatize predicate extensions of subintuitionistic logics in the style of traditional sequent calculus as found in [11,13,14]. There are also algebraic approaches to subintuitionistic logics using properties of distributive lattices [1,4,16]. However, those studies are limited to the propositional case, and completeness for predicate extensions of subintuitionistic logics has not been obtained by algebraic methods. On the other hand, our approach based on structured sequents can produce requirements, like Lemmas 5 and 8, for completeness of formal systems for various propositional and predicate logics defined through Kripke models. It might be possible to use other formalisms with structured sequents, e.g. display logic [2,19], for axiomatizing predicate extensions of subintuitionistic logics (see also discussions in [8]). Making precise connections between such formalisms and ours is to be investigated and left as future work.

Acknowledgements. We are grateful to Ryo Kashima for giving us introductions to tree-sequent calculus and Ichiro Hasuo for advice on drawing diagrams of tree-sequents. We also thank anonymous referees for valuable comments. This work was partially supported by the Japanese Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Young Scientists (B) 17700003.

References

1. Ardeshir, M., Ruitenburg, W.: Basic Propositional Calculus I. Math. Logic Quart 44, 317–343 (1998)
2. Belnap, N.D.: Display logic. J. Philos. Logic 11, 375–417 (1982)
3. Celani, S., Jansana, R.: A closer look at some subintuitionistic logics. Notre Dame J. Formal Logic 42, 225–255 (2001)
4. Celani, S., Jansana, R.: Bounded distributive lattices with strict implication. Math. Logic Quart 51, 219–246 (2005)
5. Corsi, G.: Weak logics with strict implication. Z. Math. Logik Grundlag. Math 33, 389–406 (1987)
6. Došen, K.: Modal translations in K and D. In: de Rijke, M. (ed.) Diamonds and Defaults, pp. 103–127. Kluwer Academic Publishers, Boston (1993)
7. Gabbay, D.M.: Labelled Deductive Systems. Oxford University Press, New York (1996)
8. Gabbay, D.M., Olivetti, N.: Algorithmic proof methods and cut elimination for implicational logics I: Modal implication. Studia Logica 61, 237–280 (1998)
9. Hasuo, I., Kashima, R.: Kripke completeness of first-order constructive logics with strong negation. Log. J. IGPL 11, 615–646 (2003)
10. Ishigaki, R., Kikuchi, K.: A tree-sequent calculus for a natural predicate extension of Visser's propositional logic. To appear in Log. J. IGPL
11. Ishii, K., Kashima, R., Kikuchi, K.: Sequent calculi for Visser's propositional logics. Notre Dame J. Formal Logic 42, 1–22 (2001)
12. Kashima, R.: Sequent calculi of non-classical logics — Proofs of completeness theorems by sequent calculi (in Japanese). In: Proceedings of Mathematical Society of Japan Annual Colloquium of Foundations of Mathematics, pp. 49–67 (1999)

13. Kikuchi, K.: Dual-context sequent calculus and strict implication. Math. Logic Quart 48, 87–92 (2002)
14. Kikuchi, K., Sasaki, K.: A cut-free Gentzen formulation of Basic Propositional Calculus. J. Logic Lang. Inform 12, 213–225 (2003)
15. Restall, G.: Subintuitionistic logics. Notre Dame J. Formal Logic 35, 116–129 (1994)
16. Suzuki, Y., Wolter, F., Zakharyaschev, M.: Speaking about transitive frames in propositional languages. J. Logic Lang. Inform 7, 317–339 (1998)
17. Tanaka, Y.: Cut-elimination theorems for some infinitary modal logics. Math. Logic Quart 47, 327–339 (2001)
18. Visser, A.: A propositional logic with explicit fixed points. Studia Logica 40, 155–175 (1981)
19. Wansing, H.: Displaying as temporalizing, sequent systems for subintuitionistic logics. In: Akama, S. (ed.) Logic, Language and Computation, pp. 159–178. Kluwer Academic Publishers, Boston (1997)
20. Zimmermann, E.: A predicate logical extension of a subintuitionistic propositional logic. Studia Logica 72, 401–410 (2002)

A Sequent Calculus for Bilattice-Based Logic and Its Many-Sorted Representation

Ekaterina Komendantskaya*

Department of Mathematics,
University College Cork,
Ireland
e.komendantskaya@mars.ucc.ie

Abstract. We introduce a sequent calculus for bilattice-based annotated logic (BAL). We show that this logic can be syntactically and semantically translated into a fragment MSL* of conventional many-sorted logic MSL. We show deductive equivalence of sequent calculus for BAL and sequent calculus for MSL*.

1 Introduction

Lattice and bilattice-based logics were originally introduced by several independent authors, such as Kifer, Lozinskii, Subrahmanian [15,16], Fitting [7,8,9]; Lu, Murray and Rosenthal [20,21,22] and were further developed in [13,24,25]. See also [14,1] for a very good survey of many-valued proof procedures. These logics were seen as a formalism suitable for automated reasoning about uncertainty. Most of the mentioned approaches made use of the so-called annotated, or signed, languages, where elements of a set of truth values are allowed in the syntax of the language. We will follow the approach of [15,16,13,25] and attach annotations to first-order formulae, such that a typical annotated atom of the language will be of the form $A : \mu$, where A is a first-order atom, and μ represents some annotation taken from a set of truth-values. Complex formulae can be built using these atoms.

We propose a sequent calculus for a first-order bilattice-based annotated logic (BAL). Although the papers we mentioned above provided us with many interesting insights and techniques, the calculus we define here is the first sequent calculus for a full fragment of bilattice-based annotated logics we know of.

Syntactically, we follow [16] and allow not only constant annotations, as in signed logics ([20,21,13,24,25]), but also annotation variables and terms. We develop the annotated syntax of [16] and enrich it by new bilattice connectives and quantifiers and also we allow annotations over complex first-order formulae, which is a reasonable option for annotated logics.

* I am grateful to the anonymous referees for their useful comments and suggestions. I thank the grant "Categorical Semantics for Natural Methods of Computation" by the Royal Society/Royal Irish Academy.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 165–182, 2007.

Semantically, we work with arbitrary distributive bilattices defined in [10,11]. We believe that a one-lattice fragment of BAL can be described, with some syntactical restrictions, in terms of lattice-based annotated logics [13,15,16,24,25]. However, the second lattice constituting the underlying bilattice of BAL determines some novel and non-trivial properties of its models and sequent calculus, see Proposition 2.

The many-valued sequent calculus for BAL was not our primary goal, though. In the second part of the paper we use it for a more general purpose of receiving a conventional syntactical and semantical representation of annotated many-valued logics. Annotated logics are being criticised for allowing semantics interfere with their syntax. Because of their complicated syntax, these logics have not yet received a sufficiently general structural or categorical characterisation, see also [18] for some discussion of this.

Here we continue the work of Manzano [23] translating different non-classical logics into conventional many-sorted logic. We show here that BAL can be semantically, syntactically, and deductively translated into many-sorted logic of [4,23]. The method of translation can be applied, with minor modifications, to any many-valued annotated logic we mentioned above. Moreover, our results, together with well-known results of Manzano [23] on translation of dynamic (modal) logic and second-order logic into many-sorted logic, include many-valued logics into the family of non-classical logics representable in many-sorted logic. This enables us to compare some properties of (bi)lattice-based logics and the non-classical logics just mentioned.

The structure of the paper is as follows. In Section 2, we define bilattices following [11,10]. In Section 3, we describe Bilattice-Based Annotated Logic (BAL) and sequent calculus for it. In Section 4, we define a fragment MSL* of the many-sorted logic MSL, which we use for translation of BAL. In the same section, we show the syntactical, semantical and deductive equivalence of BAL and MSL*. We summarise the results of the previous sections and outline the future work in Section 5.

2 Bilattices

In this section we briefly discuss bilattices and their properties, which will be useful in later sections, where we define bilattice-based language and theory. Notion of a bilattice generalises the Belnap's lattice with four values - *true*, *false*, *none*, *both*. This lattice of Belnap suggested that we can compare facts not only from the point of view of them being true or false, but we can also question how much information about facts is available to us. This gave rise to the notion of a bilattice - a structure with two orderings, usually called *truth* and *knowledge* ordering.

We use the well-known definition of bilattices due to Ginsberg, see [11] or [10].

Definition 1. *A* bilattice $\mathbf{B}$ *is a sextuple* $(\mathbf{B}, \vee, \wedge, \oplus, \otimes, \neg)$ *such that* $(\mathbf{B}, \vee, \wedge)$ *and* $(\mathbf{B}, \oplus, \otimes)$ *are both complete lattices, and* $\neg : \mathbf{B} \rightarrow \mathbf{B}$ *is a mapping satisfying the following three properties:* $\neg^2 = Id_{\mathbf{B}}$*,* $\neg$ *is a dual lattice homomorphism from* $(\mathbf{B}, \vee, \wedge)$ *to* $(\mathbf{B}, \wedge, \vee)$*, and* $\neg$ *is the identity mapping on* $(\mathbf{B}, \oplus, \otimes)$*.*

Let $L_1 = (\mathcal{L}_1, \leq_1)$ and $L_2 = (\mathcal{L}_2, \leq_2)$ be two lattices, let x_1, x_2 denote arbitrary elements of the lattice L_1, and let y_1, y_2 denote arbitrary elements of the lattice L_2. Let $\cap_1, \cup_1$ denote the meet and join defined in the lattice L_1, and let $\cap_2, \cup_2$ denote the meet and join defined in the lattice L_2.

Definition 2. *Suppose $L_1 = (\mathcal{L}_1, \leq_1)$ and $L_2 = (\mathcal{L}_2, \leq_2)$ are complete lattices. Form the set of points $\mathcal{L}_1 \times \mathcal{L}_2$, and define the two orderings $\leq_t$ and $\leq_k$ on $\mathcal{L}_1 \times \mathcal{L}_2$ as follows.*
(1) $\langle x_1, y_1\rangle \leq_t \langle x_2, y_2\rangle$ if and only if $x_1 \leq_1 x_2$ and $y_2 \leq_2 y_1$.
(2) $\langle x_1, y_1\rangle \leq_k \langle x_2, y_2\rangle$ if and only if $x_1 \leq_1 x_2$ and $y_1 \leq_2 y_2$.

Therefore, $\langle x_1, y_1\rangle \wedge \langle x_2, y_2\rangle = \langle x_1 \cap_1 x_2, y_1 \cup_2 y_2\rangle$, $\langle x_1, y_1\rangle \vee \langle x_2, y_2\rangle = \langle x_1 \cup_1 x_2, y_1 \cap_2 y_2\rangle$, $\langle x_1, y_1\rangle \otimes \langle x_2, y_2\rangle = \langle x_1 \cap_1 x_2, y_1 \cap_2 y_2\rangle$, and $\langle x_1, y_1\rangle \oplus \langle x_2, y_2\rangle = \langle x_1 \cup_1 x_2, y_1 \cup_2 y_2\rangle$.

We denote the resulting structure by $L_1 \times L_2 = (\mathcal{L}_1 \times \mathcal{L}_2, \leq_t, \leq_k) = (\mathbf{B}, \leq_t, \leq_k)$, where $\mathbf{B}$ denotes $\mathcal{L}_1 \times \mathcal{L}_2$.

Infinite meet and join with respect to k- and t-orderings will be denoted by $\prod$, $\sum$, $\bigwedge$, $\bigvee$.

From the definition of a lattice, it follows that $\oplus$, $\otimes$, $\vee$ and $\wedge$ are idempotent, commutative, associative, and the absorption laws hold for $\oplus$ and $\otimes$, and $\vee$ and $\wedge$. See, for example, [12] for more details.

The unary operation $\neg$, the complement with respect to the truth ordering, is defined in $L_1 \times L_2$ as follows: $\neg\langle x, y\rangle = \langle y, x\rangle$. This particular definition of negation requires that $L_1 = L_2$, and so we make this assumption throughout. There are some alternative definitions of negation in bilattices, [6,10], but we do not use them here.

Proposition 1. *[5,11] Suppose $\mathbf{B}$ is a distributive bilattice. Then there are distributive lattices L_1 and L_2 such that $\mathbf{B}$ is isomorphic to $L_1 \times L_2$.*

Therefore, we will consider only logic programs over distributive bilattices and regard the underlying bilattice of any program as a product of two lattices. Moreover, we always treat each bilattice we work with as isomorphic to some finite subset of $\mathbf{B} = L_1 \times L_2 = ([0,1], \leq) \times ([0,1], \leq)$, where $[0,1]$ is the unit interval of reals with the linear ordering defined on it. Elements of such a bilattice are pairs: the first element of each pair denotes evidence for a fact, and the second element denotes evidence against it. We assume that each bilattice we work with has four extreme elements, $\langle 0,0\rangle$ (*none*), $\langle 0,1\rangle$ (*false*), $\langle 1,0\rangle$ (*true*) and $\langle 1,1\rangle$ (*both*).

3 Bilattice-Based Annotated Logic

Now we are ready to introduce Bilattice-Based Annotated Logic. We define its syntax, semantics, and sound and complete sequent calculus for it, see also [19].

We define the bilattice-based annotated language $\mathcal{L}$ to consist of individual variables, individual constants, functions and predicate symbols together with annotation variables, annotation constants and annotation functions over some fixed bilattice.

Individual variables, constants, and functions are used to build terms, while annotation variables, constants and functions are used to build annotation terms. Typical annotation functions will be $\oplus, \otimes, \wedge, \vee$ taken from a given bilattice.

We allow several connectives to represent operations over a bilattice, and so we allow five connectives and four quantifiers, as follows: $\oplus, \otimes, \wedge, \vee, \neg$ $\Sigma, \Pi, \exists, \forall$.

We use the conventional definition of a first-order term and a first-order formula. An *annotated formula* is defined inductively as follows: if R is an n-ary predicate symbol, $t_1, \ldots, t_n$ are terms, τ is an annotation term, then $R(t_1, \ldots, t_n) : (\tau)$ is an *annotated formula* (called an *annotated atom*). If ψ, ϕ are annotated formulae, then $\psi \odot \phi$ is an annotated formula, where $\odot$ is one of $\{\oplus, \otimes, \wedge, \vee\}$. If F_1 and F_2 are formulae, and τ is an annotation term, then $(F_1 \odot F_2) : (\tau)$ is an annotated formula, where $\odot$ is one of $\{\oplus, \otimes, \wedge, \vee\}$. If ϕ is an annotated formula, then $\neg\phi$, $\Sigma\phi, \Pi\phi, \exists\phi$ and $\forall\phi$ are annotated formulae. In order to reflect the nature of annotations taken from the bilattice, we will alternatively use notation (μ, ν) instead of (τ) when discussing annotation terms.

Note that we use angled brackets $\langle\ \rangle$ to denote elements of $\mathbf{B}$ and round brackets ( ) to denote annotations of the language $\mathcal{L}$.

Example 1. Consider a binary predicate `connected`, which describes the fact of existence of an edge in a probabilistic graph. These graphs can be used to describe the behaviour of internet connections, for example. Then $\texttt{connected}(a, b) : (\frac{1}{3}, \frac{2}{3})$ will describe the fact that the probability of establishing a connection between nodes a and b is equal to $\frac{1}{3}$, while probability of losing this connection is $\frac{2}{3}$. Then, for example, $\texttt{connected}(a, b) : (\frac{1}{3}, \frac{2}{3}) \wedge (\mu, \nu)$ is also a formula which contains a function $\wedge$ and two free variables (μ, ν) in its annotation.

3.1 Interpretations

Let $\mathbf{B}$ denote a bilattice underlying the annotated language $\mathcal{L}$; it will provide the set of truth values for $\mathcal{L}$.

Following the conventional definition of an interpretation (see [3], for example), we fix a domain D, where constants and function symbols receive interpretation. A *variable assignment* V is an assignment, to each variable in $\mathcal{L}$, of an element in the domain D. An interpretation of constants, variables, and function symbols in D will be called a pre-interpretation, we will denote it by $|\ |$. An *interpretation* $\mathcal{I}$ for a (first-order) bilattice-based annotated language $\mathcal{L}$ consists of a pre-interpretation together with a mapping $|R|_{\mathcal{I}} : D^n \longrightarrow \mathbf{B}$ for each n-ary predicate symbol R in $\mathcal{L}$.

One further piece of notation we need is as follows: for each element $\langle\alpha, \beta\rangle$ of $\mathbf{B}$, we denote by $\chi_{\langle\alpha,\beta\rangle} : \mathbf{B} \longrightarrow \mathbf{B}$ the mapping defined by $\chi_{\langle\alpha,\beta\rangle}(\langle\alpha', \beta'\rangle) = \langle 1, 0\rangle$ if $\langle\alpha, \beta\rangle \leq_k \langle\alpha', \beta'\rangle$ and $\chi_{\langle\alpha,\beta\rangle}(\langle\alpha', \beta'\rangle) = \langle 0, 1\rangle$ otherwise.

We will use the two functions $\mathcal{I}$ and χ to define interpretation I for annotated atoms. Given an annotated atom $A : (\alpha', \beta')$ with constant annotation (α', β'), an interpretation $\mathcal{I}$ for a first-order formula A, and a value $\langle\alpha, \beta\rangle$ from $\mathbf{B}$ assigned to A, we use χ as follows: if the value $\langle \alpha, \beta \rangle \geq_k \langle \alpha', \beta' \rangle$, then $I(A : (\alpha', \beta')) = \langle 1, 0 \rangle$, and $I(A : (\alpha', \beta')) = \langle 0, 1 \rangle$ otherwise. If the annotated term τ attached to an annotated atom $A : \tau$ contains variables $\overline{\mu}, \overline{\nu}$, we use existential quantifier Σ when applying χ as follows: $\chi_\tau(\langle \alpha, \beta \rangle) = \langle 1, 0 \rangle$ if $\Sigma(\overline{\mu}, \overline{\nu})(\tau \geq \langle \alpha, \beta \rangle)$. We will assume this quantification when giving the next definition.

Definition 3. *Let $\mathcal{I}$ be a (first-order) interpretation with domain D for a first order annotated language $\mathcal{L}$ and let V be a variable assignment. Then an annotated formula F in $\mathcal{L}$ can be given an interpretation $I(F)$ (also denoted as $|F|_{I,V}$) in* **B** *as follows.*

- *If F is an annotated atom $R(t_1, \ldots, t_n) : (\mu, \nu)$ with (μ, ν) being an annotation term, then the value of $|F|_{I,V}$ is given by $|F|_{I,V} = \chi_{\langle \mu, \nu \rangle}$ $(|R|_{\mathcal{I}}(|t_1|, \ldots, |t_n|))$.*
- *If F has the form $(\neg A) : (\mu, \nu)$, with A being some first-order atom and (μ, ν) being an annotation term, then $|(\neg A) : (\mu, \nu)|_{I,V} = |A : (\nu, \mu)|_{I,V}$.*
- *If F has the form $\neg(A : (\mu, \nu))$, with A being some first-order formula and (μ, ν) being an annotation term, then $|\neg(A : (\mu, \nu))|_{I,V} = \neg^*(|A : (\mu, \nu)|_{I,V})$, where the operation $\neg^*$ denotes the restriction of the bilattice operation $\neg$ to the set of values $\{\langle 1, 0 \rangle, \langle 0, 1 \rangle\}$.*
- *If F has the form $(F_1 \otimes F_2)$, where F_1 and F_2 are annotated atoms, then $|F_1 \otimes F_2|_{I,V} = |F_1|_{I,V} \otimes |F_2|_{I,V}$.*

  Note that on the left hand side of this equation, the symbol $\otimes$ denotes a connective in $\mathcal{L}$, and on the right hand side it denotes an operation of the bilattice **B**.
- *If an annotated formula has the form $(A_1 \otimes A_2) : (\mu, \nu)$, where A_1, A_2 are first-order formulae and (μ, ν) being an annotation term, then $|(A_1 \otimes A_2) : (\mu, \nu)|_{I,V} = \chi_{\langle \mu, \nu \rangle}(|A_1|_{\mathcal{I},V} \otimes |A_2|_{\mathcal{I},V})$.*
- *If a formula has the form $\Sigma x R(x) : (\mu, \nu)$, with (μ, ν) being an annotation term, then*

$$|\Sigma x R(x) : (\mu, \nu)|_{I,V} = \chi_{\langle \mu, \nu \rangle} \left(\sum_{d \in D} |R(d)|_{\mathcal{I},V} \right),$$

  where $\sum$ is the infinite join with respect to $\leq_k$, $|R(d)|_{\mathcal{I},V}$ receives interpretation with respect to $\mathcal{I}$ and $V(\frac{x}{d})$, where $V(\frac{x}{d})$ is V except that x is assigned d.

We omit the definitions of interpretations for the remaining connectives and quantifiers, and the reader can easily complete Definition 3. We simply mention here that the connectives $\oplus$, $\otimes$, $\wedge$, $\vee$ and the quantifiers Σ, Π, $\exists$, $\forall$ are interpreted by finite and infinite operations defined on bilattices as in Section 2.

In general, the analogs of the classical truth values true and false are represented by $\langle 1, 0 \rangle$ and $\langle 0, 1 \rangle$ - the greatest and least elements of the bilattice with respect to $\leq_t$. Furthermore, the definitions of satisfiable, unsatisfiable, valid and non-valid formulae and of models are standard if the classical truth values true and false in these definitions are replaced by $\langle 1, 0 \rangle$ and $\langle 0, 1 \rangle$.

The following properties of I are very important for the development of a sequent calculus for BAL.

Proposition 2 (Properties of I). *Let $F, F_1, \dots F_k$ be first-order formulae, and fix a first-order interpretation $\mathcal{I}$ for them. Then any interpretation I built upon $\mathcal{I}$ as in Definition 3 has the following properties:*

1. If $I(F : (\alpha, \beta)) = \langle 1, 0\rangle$, then $I(F : (\alpha', \beta')) = \langle 1, 0\rangle$ for all $\langle \alpha', \beta'\rangle \leq_k \langle \alpha, \beta\rangle$.

2. If $I(F : (\alpha, \beta)) = \langle 0, 1\rangle$, then $I(F : (\alpha', \beta')) = \langle 0, 1\rangle$, for all (α', β') such that $\langle \alpha, \beta\rangle \leq_k \langle \alpha', \beta'\rangle$.

3. $I(F_1 : (\mu_1, \nu_1) \otimes \dots \otimes F_k : (\mu_k, \nu_k)) = \langle 1, 0\rangle \iff I(F_1 : (\mu_1, \nu_1) \oplus \dots \oplus F_k : (\mu_k, \nu_k)) = \langle 1, 0\rangle \iff I(F_1 : (\mu_1, \nu_1) \wedge \dots \wedge F_k : (\mu_k, \nu_k)) = \langle 1, 0\rangle \iff I(F_i : (\mu_i, \nu_i)) = \langle 1, 0\rangle$, for each $i \in \{1, \dots, k\}$.

4. $I(F_1 : (\mu_1, \nu_1) \otimes \dots \otimes F_k : (\mu_k, \nu_k)) = \langle 0, 1\rangle \iff I(F_1 : (\mu_1, \nu_1) \oplus \dots \oplus F_k : (\mu_k, \nu_k)) = \langle 0, 1\rangle \iff I(F_1 : (\mu_1, \nu_1) \vee \dots \vee F_k : (\mu_k, \nu_k)) = \langle 0, 1\rangle \iff I(F_i : (\mu_i, \nu_i)) = \langle 0, 1\rangle$, for each $i \in \{1, \dots, k\}$.

5. If $I(F_1 : (\mu_1, \nu_1) \odot \dots \odot F_k : (\mu_k, \nu_k)) = \langle 1, 0\rangle$, then $I((F_1 \odot \dots \odot F_k) : ((\mu_1, \nu_1) \odot \dots \odot (\mu_k, \nu_k))) = \langle 1, 0\rangle$, where $\odot$ is any one of the connectives $\otimes, \oplus, \wedge$.

6. If $I(F_1 : (\mu_1, \nu_1) \odot \dots \odot F_k : (\mu_k, \nu_k)) = \langle 0, 1\rangle$, then $I((F_1 \odot \dots \odot F_k) : ((\mu_1, \nu_1) \odot \dots \odot (\mu_k, \nu_k))) = \langle 0, 1\rangle$, where $\odot$ is any one of the connectives $\otimes, \oplus, \vee$.

7. If $I(F_1 : (\mu, \nu)) = \langle 1, 0\rangle$, then $I((F_1 \oplus F_2) : (\mu, \nu)) = \langle 1, 0\rangle$, for any formula F_2.

8. If $I(F_1 : (\mu, \nu)) = \langle 0, 1\rangle$, then $I((F_1 \otimes F_2) : (\mu, \nu)) = \langle 0, 1\rangle$, for any formula F_2.

9. $I(F : (\mu, \nu)) = \langle 1, 0\rangle \iff I(\neg F : (\nu, \mu)) = \langle 1, 0\rangle$.

10. $I(F : (\mu, \nu)) = \langle 0, 1\rangle \iff I(\neg F : (\nu, \mu)) = \langle 0, 1\rangle$.

11. For every formula F, $I(F : (0, 0)) = \langle 1, 0\rangle$.

The proof of this proposition uses the definitions of I and bilattice operations. Not all the statements in the above proposition are immediate. For example, items $3 - 6$ exhibit some non-trivial properties of the bilattice interpretation. The proposition will be useful in the next subsection, where we introduce a sound and complete sequent calculus for BAL.

3.2 Sequent Calculus for Annotated Logics

In this section we will use properties of bilattices and interpretation I to define a sequent calculus for Bilattice-Based Annotated Logic (BAL).

We denote annotated formulae by φ, ψ and ϕ, and conventional first-order formulae by F_1, F_2. The symbols Ω and Γ will denote arbitrary finite sequences of annotated formulae.

We allow the following axioms $\varphi \mapsto \varphi$; $\mapsto F : (0, 0)$ and rules:

1. Introducing $\neg$:

$$\frac{\Omega \mapsto \Gamma, \varphi}{\neg\varphi, \Omega \mapsto \Gamma,}\ \neg\text{L}\ , \qquad \frac{\varphi, \Omega \mapsto \Gamma}{\Omega \mapsto \Gamma, \neg\varphi}\ \neg\text{R}.$$

2. Introducing $\vee$:

$$\frac{\psi, \Omega \mapsto \Gamma;\ \phi, \Omega \mapsto \Gamma}{\phi \vee \psi, \Omega \mapsto \Gamma}\ \vee\text{-L}, \qquad \frac{\Omega \mapsto \Gamma, \varphi}{\Omega \mapsto \Gamma, \varphi \vee \psi} \text{ or } \frac{\Omega \mapsto \Gamma, \psi}{\Omega \mapsto \Gamma, \varphi \vee \psi}\ \vee\text{-R}.$$

3. Introducing $\wedge$:

$$\frac{\psi, \Omega \mapsto \Gamma}{\psi \wedge \phi, \Omega \mapsto \Gamma} \text{ or } \frac{\phi, \Omega \mapsto \Gamma}{\psi \wedge \phi, \Omega \mapsto \Gamma} \wedge\text{-L}, \qquad \frac{\Omega \mapsto \Gamma, \varphi \ ; \ \Omega \mapsto \Gamma, \psi}{\Omega \mapsto \Gamma, \varphi \wedge \psi} \wedge\text{-R.}$$

4. Introducing $\oplus$:

$$\frac{\psi, \Omega \mapsto \Gamma; \ \phi, \Omega \mapsto \Gamma}{\psi \oplus \phi, \Omega \mapsto \Gamma} \oplus\text{-L}, \qquad \frac{\Omega \mapsto \Gamma, \varphi \ ; \ \Omega \mapsto \Gamma, \psi}{\Omega \mapsto \Gamma, \varphi \oplus \psi} \oplus\text{-R.}$$

The rule $\oplus$-L is identical to $\vee$-L; but the rule $\oplus$-R is identical to $\wedge$-R. Cf. Definition 2 and Proposition 2.(3-4).

5. Introducing $\otimes$:

$$\frac{\psi, \Omega \mapsto \Gamma; \ \varphi, \Omega \mapsto \Gamma}{\psi \otimes \varphi, \Omega \mapsto \Gamma} \otimes\text{-L}, \qquad \frac{\Omega \mapsto \Gamma, \varphi \ ; \ \Omega \mapsto \Gamma, \psi}{\Omega \mapsto \Gamma, \varphi \otimes \psi} \otimes\text{-R.}$$

The rule $\otimes$-R is identical to $\wedge$-R; but $\otimes$-L is identical to $\vee$-L. Cf. Definition 2 and Proposition 2.(3-4).

Similarly, because quantifiers Π, Σ are modelled by infinite analogues of $\otimes$ and $\oplus$, rules for introducing Π are identical to the rules for Σ:

6. Introducing $\bigcirc = \Sigma, \Pi$ in the antecedent (Σ-L/ Π-L):

$$\frac{S^{y_i}_{x_i}\varphi, \Omega, \mapsto \Gamma}{\bigcirc x_i \varphi, \Omega \mapsto \Gamma}, y_i \notin FREE(\Omega \cup \{\Sigma x_i \varphi, \Gamma\}).$$

7. Introducing $\bigcirc = \Sigma, \Pi$ in the consequent (Σ-R/ Π-R):

$$\frac{\Omega \mapsto \Gamma, S^{y_i}_{x_i}\varphi}{\Omega \mapsto \Gamma, \bigcirc x_i \varphi}, y_i \notin FREE(\Omega \cup \{\Sigma x_i \varphi, \Gamma\}).$$

10. Introducing $\exists$ in the antecedent ($\exists$-L):

$$\frac{S^{y_i}_{x_i}\varphi, \Omega \mapsto \Gamma}{\exists x_i \varphi, \Omega \mapsto \Gamma}, y_i \notin FREE(\Omega \cup \{\exists x_i \varphi, \Gamma\}).$$

11. Introducing $\exists$ in the consequent ($\exists$-R):

$$\frac{\Omega \mapsto \Gamma, S^{t}_{x_i}\varphi}{\Omega \mapsto \Gamma, \exists x_i \varphi}.$$

The rules for quantifier $\forall$ are given dually to those for $\exists$, as in conventional classical sequent calculi.

The next block of rules will give an account of annotations:

12. Increasing values in annotations (IA), (*cf. Proposition 2.(1)*):

$$\frac{\Omega \mapsto \Gamma, F : (\mu, \nu)}{\Omega \mapsto \Gamma, F : (\mu', \nu')}, \text{ for all } (\mu', \nu') \leq_k (\mu, \nu).$$

13. Descending values in annotations (DA) (*cf. Proposition 2.(2)*):

$$\frac{F : (\mu, \nu), \Omega \mapsto \Gamma}{F : (\mu', \nu'), \Omega \mapsto \Gamma}, \text{ for all } (\mu, \nu) \leq_k (\mu', \nu').$$

14. Introducing $\oplus$ inside annotated formulae in the antecedent (I$\oplus$I -L) (*cf. Proposition 2.(7)*):

$$\frac{F_1 : (\mu, \nu), \Omega \mapsto \Gamma;\ F_2 : (\mu, \nu), \Omega \mapsto \Gamma}{(F_1 \oplus F_2) : (\mu, \nu), \Omega \mapsto \Gamma}\ .$$

15. Introducing $\oplus$ inside annotated formulae in the consequent ($\oplus$I-R) (*cf. Proposition 2.(7)*):

$$\frac{\Omega \mapsto \Gamma, F_1 : (\mu, \nu)}{\Omega \mapsto \Gamma, (F_1 \oplus F_2) : (\mu, \nu)}\ , \qquad \frac{\Omega \mapsto \Gamma, F_2 : (\mu, \nu)}{\Omega \mapsto \Gamma, (F_1 \oplus F_2) : (\mu, \nu)}\ .$$

16. Introducing $\otimes$ inside annotated formulae in the antecedent ($\otimes$I-L) (*cf. Proposition 2.(8)*):

$$\frac{F_1 : (\mu, \nu), \Omega \mapsto \Gamma}{(F_1 \otimes F_2) : (\mu, \nu), \Omega \mapsto \Gamma}\ , \qquad \frac{F_2 : (\mu, \nu), \Omega \mapsto \Gamma}{(F_1 \otimes F_2) : (\mu, \nu), \Omega \mapsto \Gamma}\ .$$

17. Introducing $\otimes$ inside annotated formulae in the consequent (I $\otimes$ I-R) (*cf. Proposition 2.(8)*):

$$\frac{\Omega \mapsto \Gamma, F_1 : (\mu, \nu);\ \Omega \mapsto \Gamma, F_2 : (\mu, \nu)}{\Omega \mapsto \Gamma, (F_1 \otimes F_2) : (\mu, \nu)}\ .$$

18. Combining annotations in the antecedent (CAL) (*cf. Proposition 2.(6)*):

$$\frac{F_1 : (\mu_1, \nu_1) \odot F_2 : (\mu_2, \nu_2), \Omega \mapsto \Gamma}{(F_1 \odot F_2) : ((\mu_1, \nu_1) \odot (\mu_2, \nu_2)), \Omega \mapsto \Gamma}, \text{ where } \odot \text{ is either } \oplus, \otimes \text{ or } \vee.$$

19. Combining annotations in the consequent (CAR) (*cf. Proposition 2.(5)*, see also Example 2):

$$\frac{\Omega \mapsto \Gamma, F_1 : (\mu_1, \nu_1) \odot F_2 : (\mu_2, \nu_2)}{\Omega \mapsto \Gamma, (F_1 \odot F_2) : ((\mu_1, \nu_1) \odot (\mu_2, \nu_2))}, \text{ where } \odot \text{ is either } \oplus, \otimes \text{ or } \wedge.$$

20. Introducing $\neg$ inside annotations (*cf. Proposition 2.(11)*):

$$\frac{F_1 : (\nu, \mu), \Omega \mapsto \Gamma}{(\neg F_1) : (\mu, \nu), \Omega \mapsto \Gamma}\neg\text{I-L}\ , \qquad \frac{\Omega, \mapsto \Gamma, F_1 : (\nu, \mu)}{\Omega \mapsto \Gamma, (\neg F_1) : (\mu, \nu)}\neg\text{I-R}\ .$$

We allow the structural rules interchange, contraction and weakening. These structural rules can be defined to be either primitive or admissible, in style of **G3**. The latter option seems to be more appropriate for automated reasoning, but we shall not discuss this issue here.

All the annotation functions $\otimes$, $\oplus$, $\wedge$ and $\vee$ are defined in **B**, and one is allowed to operate with them accordingly. That is, for example, one can think of $F : ((\frac{1}{3}, \frac{2}{3}) \otimes (\frac{2}{3}, \frac{1}{3}))$ as of $F : (\frac{1}{3}, \frac{1}{3})$. Also, because in both lattices constituting a bilattice, operations $\oplus$, $\otimes$, $\wedge$ and $\vee$ are idempotent, commutative, associative, and distributive, one can treat equally $F : (\mu, \nu)$ and $F : ((\mu, \nu) \odot (\mu, \nu))$ ($\odot = \oplus, \otimes, \wedge, \vee$), for example, or write $F : ((\mu_1, \nu_1) \vee ((\mu_2, \nu_2) \wedge (\mu_3, \nu_3)))$ instead of $F : (((\mu_1, \nu_1) \vee (\mu_2, \nu_2)) \wedge ((\mu_1, \nu_1) \vee (\mu_3, \nu_3)))$, and so on. All these properties can be stated directly as sequent rules, or, as we do here, just assumed throughout. In fact, the latter way seems to be more natural: $\otimes, \oplus, \wedge$ and $\vee$ appearing in annotations are not connectives, but they are annotation functions. And, in the same way as we ignore properties of functions appearing in individual terms when defining conventional sequent rules, we may wish to ignore properties of annotation functions when defining sequent calculus for BAL.

In general, all the classical tautologies reformulated with annotated formulae are provable in this calculus. But additionally, we can prove theorems concerning properties of annotations.

Example 2. The formula $(F_1 \wedge F_2) : (0, 1)$ is a logical consequence of $F_1 : (1, 0) \wedge F_2 : (0, 1)$. One can use the rule CAR and the fact that annotation terms $(1, 0) \wedge (0, 1)$ and $(0, 1)$ are equal to prove this sequentially.

Cut elimination theorem can be proven for BAL, if cut rule is defined. But we will not discuss this issue here.

Theorem 1 (Soundness). *For any annotated formula φ, if $\psi \vdash \varphi$ then $\psi \models \varphi$.*

Proof. Proof follows along the lines of conventional proof of soundness for classical first-order sequent calculus. We additionally make use of Proposition 2.

The calculus for BAL is also complete. But we avoid to state Completeness until the last section, when we obtain it as a corollary of the completeness theorem for many-sorted logic.

4 Many-Sorted Representation of BAL

The sequent calculus for BAL introduced in the previous section reflects rigorously the semantic properties of the logic. But it may be criticised for having a difficult rule representation and allowing semantics to interfere with its syntax. In the second part of this paper we will show how this calculus can equivalently be transformed into the elegant conventional sequent calculus for many-sorted logic, introduced in [4]. The latter is proven to be sound and complete, see, for example [4,23].

In this section we draw inspiration from the work of Manzano [23] who showed how second-order and dynamic logic can be translated into fragments of many-sorted logic MSL. We define the syntax and semantics of its fragment MSL* and use it for translation of BAL.

We follow the notation of Manzano when working with many-sorted semantics. We hope that the uniformity of our notation with that of [23] will make it easier to consider our results in one context with the similar results of Manzano concerning second-order and dynamic logic.

4.1 Many-Sorted Language MSL* of Signature Σ^*

We start by defining the many-sorted language MSL* of signature Σ^*.

Definition 4. *We define a* signature $\Sigma^* = \langle \mathrm{SORT}, \mathrm{RANK} \rangle$, *where*

$\mathrm{SORT}(\Sigma^*) = \mathrm{SORT} = \{0, 1, 2, < 0, \overbrace{1, \ldots, 1}^{n-1}, 2 >\}$ *(representing boolean, individual, bilattice universes and a universe of n-ary relations on individuals and bilattice elements); and* RANK *is described in Df. 5.*

We will continue defining RANK in the next Definition, and will pause for a while to explain the significance of the particular choice of SORT for MSL*. The sorts 2 and $< 0, \overbrace{1, \ldots, 1}^{n-1}, 2 >$ are specific for MSL*, if we compare them with the general definition of a many-sorted signature in [23]. The sort 2 represents universe of bilattice elements. The sort $< 0, \overbrace{1, \ldots, 1}^{n-1}, 2 >$ will be used when we give a formal account of the interpretation function I for BAL, which will be formalised by relations I^k of different arities in MSL*. As in second-order logic, the relations interpreted in this universe will be quantified. Moreover, they will have both individual and annotation terms as arguments. As any other relations, they can be evaluated as true, if some elements in the underlying structure satisfy this relation, or false otherwise.

We define $S_\omega(\mathrm{SORT})$ to be the set of all finite sequences of elements of SORT.

RANK is a function whose values are in $S_\omega(\mathrm{SORT})$.

We denote Dom(RANK) as OPER.SYM(Σ^*) = OPER.SYM and call its elements operation symbols. We allow the following operation symbols in the language:

- $\neg, \vee, \wedge$ - the classical connectives;
- $f_1^n, f_2^n, \ldots$, for any $n \in \mathbf{N}$, - function symbols of different arities over individual terms;
- $\vartheta_1^n, \vartheta_2^n, \ldots$, for any $n \in \mathbf{N}$, - bilattice function symbols of different arities;
- $R_1^n, R_2^n, \ldots$, for any $n \in \mathbf{N}$, - predicates over individual terms;
- $\leq_k$ - bilattice binary relation;
- $I_1^n, I_2^n, \ldots,$, for any $n \in \mathbf{N}$, - relation symbols of different arities;
- ε - membership relation.

Then RANK is defined for each of them as follows.

Definition 5 (Df. 4 continued).
For $\neg, \vee, \wedge \in$ OPER.SYM, $\mathrm{RANK}(\vee) = \mathrm{RANK}(\wedge) = \langle 0, 0, 0 \rangle$,
$\mathrm{RANK}(\neg) = \langle 0, 0 \rangle$.

We define $\mathrm{RANK}(f^n(x_1, \ldots, x_n)) = \langle 1, \overbrace{1, \ldots, 1}^{n} \rangle$, *where f^n is n-ary function over terms of sort* 1*; and*

$\mathrm{RANK}(\vartheta^n((\mu_1, \nu_1), \ldots, (\mu_n, \nu_n))) = \langle 2, \overbrace{2, \ldots, 2}^{n} \rangle$, *where ϑ^n is n-ary function over terms of sort* 2*;*

$\mathrm{RANK}(R^n(x_1, \ldots, x_n)) = \langle 0, \overbrace{1, \ldots, 1}^{n} \rangle$, *where R^n is n-ary relation over individual terms of sort* 1*;*

For the binary bilattice relation $\leq_k$*,* $\mathrm{RANK}(\leq_k) = \langle 0, 2, 2\rangle$*.*

$\mathrm{RANK}(I^n((x_1, \ldots, x_n), (\mu, \nu))) = \langle 0, \overbrace{1, \ldots, 1}^{n}, 2\rangle$*, where* I^n *is an* n*-ary relation over terms of sorts* 1 *and* 2*.*
And, finally, for membership relations ε_n*,*

$$\mathrm{RANK}(\varepsilon_n) = \langle 0, \overbrace{1, \ldots, 1}^{n}, 2 < 0, \overbrace{1, \ldots, 1}^{n}, 2 >\rangle.$$

Note that each annotation term of the form (μ_i, ν_i) is interpreted by one element of a bilattice, and thus each annotation term has a single sort 2, and not $(2, 2)$ as one might expect.

Definition 6. *We define a many-sorted* structure

$$S = \langle A_1, A_2, A_3^n, f^{A_1}, f^{A_2}\rangle, (\textit{for each } n \in \mathbb{N}),$$

where A_1*,* A_2 *and* A_3^n *are universes for variables of sorts* $< 1 >$*,* $< 2 >$*,* $< 0, \overbrace{1, \ldots, 1}^{n-1}, 2 >$*;* $f^{A_1} \subseteq A_1^k$ *and* $f^{A_2} \subseteq A_2^k$*.*

A many-sorted language $\mathcal{L}$ consists of symbols from OPER.SYM, quantifier $\exists$ and the set of variables $\mathcal{V} = V^i : i \in \mathrm{SORT} - \{0\}$. That is, the superscript of a variable denotes its sort.

Definition 7. *Expressions of the language are defined inductively as follows:*

1. *Each variable of a sort* i *is an expression of the same type.*
2. *If* $f^m \in$ OPER.SYM *and* $\epsilon_1, \ldots, \epsilon_m$ *are expressions of the single type 1, then* $f^m(\epsilon_1, \ldots, \epsilon_m)$ *is an expression of type* 1*.*
   If $\vartheta^m \in$ OPER.SYM *and* $\epsilon_1, \ldots, \epsilon_m$ *are expressions of type* 2*, then* $\vartheta(\epsilon_1, \ldots, \epsilon_m)$ *is an expression of type* 2*.*
   If $R^m \in$ OPER.SYM *, then for all the expressions* ϵ *of the single type* 1*, the string* $R(\epsilon_1, \ldots, \epsilon_m)$ *is an expression of type* 0*.*
   If $\leq_k \in$ OPER.SYM*, then for all expressions* ϵ_1, ϵ_2 *of the single type* 2*, the string* $\epsilon_1 \leq_k \epsilon_2$ *is an expression of type* 0*.*
   If $I^{m+1} \in$ OPER.SYM*,* $\epsilon_1, \ldots, \epsilon_m$ *are expressions of the single type* 1 *and* ϵ *is an expression of the single type* 2*, then* $I^{m+1}((\epsilon_1, \ldots, \epsilon_m), \epsilon)$ *is an expression of type* 0*.*
3. *If* ϵ *is an expression of type* 0 *and* x^i *is a variable of sort* i*, then* $\exists x^i \epsilon$ *is an expression of type* 0 *as well.*

No other string is an expression.

Terms are expressions of single non-zero type $i = 1, 2$. Formulae are expressions of type 0.

Note that in our setting, the relations I of different arities can be viewed as variables and can be quantified. This is why, we handle these relations uniformly with the way how [23] treated second order relations and formulae within many-sorted language. We require $\varepsilon t_1, \ldots, t_n, \mu, \nu I^n$, with ε being the membership relation, to replace $I^n((t_1, \ldots, t_n), (\mu, \nu))$; and we define expressions of the former type to be formulae, but expressions of the latter type - not, and amend Definition 7 accordingly.

We define $\supset$ and $\forall$ using $\neg$, $\vee$ and $\exists$ in the usual way.

Interpretation. We define the interpretation function $\mathbf{I}$ for the many-sorted language, following closely [23], but making a substantial adaptation to the particular language MSL* we have defined.

We define Assignment

$$M : \bigcup \underset{i \in \mathrm{SORT}-\{0\}}{\mathcal{V}_i} \rightarrow \underset{i \in \mathrm{SORT}-\{0\}}{A_i},$$

in such a way that $M[\mathcal{V}_i] \subseteq A_i$.

Definition 8. *An interpretation* $\mathbf{I}$ *over a structure* S *is a pair* $\langle S, M \rangle$*, where* M *is an assignment on* S*. In particular, for* $i \in \mathrm{SORT}$*,*

1. $\mathbf{I}(x^i) = M(x^i)$,
2. $\mathbf{I}(f(\epsilon_1, \ldots, \epsilon_n)) = f^S(\mathbf{I}(\epsilon_1), \ldots, \mathbf{I}(\epsilon_n))$.
   As particular cases of item 2, we have:
   $\mathbf{I}(a^i) = (a^i)^S$;
3. $\mathbf{I}(f(t_1, \ldots, t_n)) = (f)^S(\mathbf{I}(t_1), \ldots, \mathbf{I}(t_n))$;
4. $\mathbf{I}(\vartheta(\tau_1, \ldots, \tau_n)) = (\vartheta)^S(\mathbf{I}(\tau_1), \ldots, \mathbf{I}(\tau_n))$;
   $\mathbf{I}(R(t_1, \ldots, t_n)) = (R)^S(\mathbf{I}(t_1), \ldots, \mathbf{I}(t_n))$;
   $\mathbf{I}((t_1, \ldots, t_n), (\tau)\varepsilon I) = \varepsilon^S \langle (\mathbf{I}(t_1), \ldots, \mathbf{I}(t_n), \mathbf{I}(\tau)), ((I)^S) \rangle$;
   $\mathbf{I}(\neg\psi) = \neg^S(\mathbf{I}(\psi))$;
   $\mathbf{I}(\psi \vee \phi) = (\mathbf{I}(\psi)) \vee^S (\mathbf{I}(\phi))$; $\mathbf{I}(\psi \wedge \phi) = (\mathbf{I}(\psi)) \wedge^S (\mathbf{I}(\phi))$.
5. $\mathbf{I}(\exists x^i \phi) =$ *True if and only if* $\{x^i \in A_i | \mathbf{I}^{x^j}_{x^i}(\phi) = True\} \neq \emptyset$ *(where* $\mathbf{I}^{x^j}_{x^i} = \langle S, M^{x^j}_{x^i} \rangle$ *and* $M^{x^j}_{x^i} = (M - \{< x^i, M(x^i) >\}) \cup \{< x^i, x^j >\}$*).*

Note that the interpretation of MSL* is two-valued: atomic formulae are interpreted as true if the relations they formalise are satisfied in the structure S, and they are interpreted false otherwise. Complex formulae are interpreted respective to this interpretation of atomic formulae.

Now, when the many-sorted language MSL*, its underlying structure, and the interpretation function $\mathbf{I}$ are defined, we will show that BAL can be translated into MSL*, and so we give syntactical and semantical translation for BAL.

4.2 Translation of BAL into MSL*

In the subsequent sections we lighten the notation, and instead of using upper indices to denote sorts of variables, will use symbols x and y with lower indices to denote variables of sort 1, and μ, ν with lower indices to denote variables of sort 2.

Syntactical Translation. The syntactical translation from BAL to MSL* leaves all individual terms and atomic non-annotated formulae as they are, and gives the following translation to the atomic annotated formulae: $\mathrm{TRANSL}_{BAL\mapsto \mathrm{MSL}^*}(R(\overline{x}) : (\mu, \nu)) =$ $\forall I \exists (\mu', \nu')((R(\overline{x}) \wedge (\overline{x}, (\mu', \nu'))\varepsilon I) \wedge ((\mu, \nu) \leq_k (\mu', \nu'))).$

We will abbreviate $\mathrm{TRANSL}_{BAL\mapsto \mathrm{MSL}^*}$ as TRANSL^*.

$$\mathrm{TRANSL}^*(\neg(R(\overline{x}) : (\mu, \nu))) = \mathrm{TRANSL}^*(R(\overline{x}) : (\nu, \mu)).$$

Let ψ_1 and ψ_2 be annotated atomic formulae. Then

$$\mathrm{TRANSL}^*(\neg\psi_1) = \neg\mathrm{TRANSL}^*(\psi_1);$$

$$\mathrm{TRANSL}^*(\psi_1 \wedge \psi_2) = \mathrm{TRANSL}^*(\psi_1) \wedge \mathrm{TRANSL}^*(\psi_2);$$

$$\mathrm{TRANSL}^*(\psi_1 \vee \psi_2) = \mathrm{TRANSL}^*(\psi_1) \vee \mathrm{TRANSL}^*(\psi_2);$$

Because our final goal is to translate sequent calculus for BAL into the conventional sequent calculus for many-sorted logics, the translation function TRANSL^* is sensitive to the position of a translated formula in a sequent:

$$\mathrm{TRANSL}^*(\psi_1 \otimes \psi_2) = \mathrm{TRANSL}^*(\psi_1) \wedge \mathrm{TRANSL}^*(\psi_2),$$

if $\psi_1 \otimes \psi_2$ appears in the consequent of a sequent; and

$$\mathrm{TRANSL}^*(\psi_1 \otimes \psi_2) = \mathrm{TRANSL}^*(\psi_1) \vee \mathrm{TRANSL}^*(\psi_2),$$

if $\psi_1 \otimes \psi_2$ appears in the consequent of a sequent.

$$\mathrm{TRANSL}^*(\psi_1 \oplus \psi_2) = \mathrm{TRANSL}^*(\psi_1) \wedge \mathrm{TRANSL}^*(\psi_2),$$

if $\psi_1 \otimes \psi_2$ appears in the antecedent of a sequent; and

$$\mathrm{TRANSL}^*(\psi_1 \oplus \psi_2) = \mathrm{TRANSL}^*(\psi_1) \vee \mathrm{TRANSL}^*(\psi_2),$$

if $\psi_1 \otimes \psi_2$ appears in the antecedent of a sequent.

For complex formulae under a single annotation we introduce the following translation

$\mathrm{TRANSL}^*(R_1(\overline{x}) \odot R_2(\overline{y}) : (\alpha, \beta)) = \forall I[\exists(\mu_1, \nu_1), (\mu_2, \nu_2)(R_1(\overline{x}) \wedge R_2(\overline{y})$ $\wedge(\overline{x}, (\mu_1, \nu_1)\varepsilon I) \wedge (\overline{y}, (\mu_2, \nu_2)\varepsilon I) \wedge ((\alpha, \beta) \leq_k ((\mu_1, \nu_1) \odot (\mu_2, \nu_2))))]$, for $\odot = \oplus, \otimes, \vee, \wedge$.

Finally, we give a translation for the existential quantifiers:

$$\mathrm{TRANSL}^*(\exists x\psi) = \exists x\mathrm{TRANSL}^*(\psi).$$

$\mathrm{TRANSL}^*(\Sigma x\psi) = \exists x\mathrm{TRANSL}^*(\psi)$, if $\Sigma x\psi$ is in the antecedent of a sequent; and $\mathrm{TRANSL}^*(\Sigma x\psi) = \forall x\mathrm{TRANSL}^*(\psi)$, if $\Sigma x\psi$ is in the consequent of a sequent.

Example 3. The ground formula $(F_1 \wedge F_2) : (0, 1)$ from Example 2 can be translated into $\forall I[\exists(\mu_1, \nu_1), (\mu_2, \nu_2)(F_1 \wedge F_2 \wedge ((\mu_1, \nu_1)\varepsilon I) \wedge ((\mu_2, \nu_2)\varepsilon I) \wedge ((0, 1) \leq_k ((\mu_1, \nu_1) \wedge (\mu_2, \nu_2))))]$.

Semantical translation. We are ready now to compare model properties of BAL and MSL*.

Lemma 1. *Let F be an annotated formula of BAL. Let Σ^* and S be a signature respectively a structure of* MSL**, with* **I** *being a many-sorted interpretation in S. And let $|\ |_I$ be an interpretation for BAL as defined in Section 3. Then the following holds:*

$$|F|_I = \langle 1,0\rangle \ \textit{in}\ \mathrm{BAL} \iff \mathbf{I}(\mathrm{TRANSL}^*(F)) = \texttt{True}\ \textit{in}\ \mathrm{MSL}^*\ .$$

Proof. The proof proceeds by two routine induction arguments on complexity of formulae in BAL and MSL*; we use definitions of I in BAL and interpretation **I** over the many-sorted structure S.

Corollary 1. $\models F$ *in* BAL *if and only if* $\models$ TRANSL$^*(F)$ *in* MSL**.*

We have given the syntactical and semantical translation of BAL into MSL*. It remains to show their deductive equivalence.

4.3 Sequent Calculus for MSL*

We define a theory which will be proven to be deductively equivalent to BAL. In fact, sequent calculus for MSL* is just a conventional many-sorted sequent calculus MSL of [4,23], but with several simple rules added in order to reflect particular properties of bilattice structures[1]:

We add $\mapsto (\mu,\nu) \geq_k (0,0)$ to the set of axioms.

The rules we add to MSL are:

1. Transitivity of $\leq_k$ in the consequent:

$$\frac{\Omega \mapsto \Gamma, (\mu_1,\nu_1) \leq_k (\mu_2,\nu_2), (\mu_2,\nu_2) \leq_k (\mu_3,\nu_3)}{\Omega \mapsto \Gamma, (\mu_1,\nu_1) \leq_k (\mu_3,\nu_3)}\ \text{Tr-R.}$$

2. Transitivity of $\leq_k$ in the antecedent:

$$\frac{(\mu_2,\nu_2) \leq_k (\mu_1,\nu_1), (\mu_3,\nu_3) \leq_k (\mu_2,\nu_2), \Omega \mapsto \Gamma}{(\mu_3,\nu_3) \leq_k (\mu_1,\nu_1), \Omega \mapsto \Gamma}\ \text{TR-L.}$$

3. Introduction of $\oplus$ in the antecedent:

$$\frac{(\mu,\nu) \leq_k (\mu_1,\nu_1), \Omega \mapsto \Gamma;\ \ (\mu,\nu) \leq_k (\mu_2,\nu_2), \Omega \mapsto \Gamma}{(\mu,\nu) \leq ((\mu_1,\nu_1) \oplus (\mu_2,\nu_2)), \Omega \mapsto \Gamma}\ \oplus\text{-L.}$$

4. Introduction of $\oplus$ in the consequent:

$$\frac{\Omega \mapsto \Gamma, (\mu,\nu) \leq_k (\mu_1,\nu_1)}{\Omega \mapsto \Gamma, (\mu,\nu) \leq_k ((\mu_1,\nu_1) \oplus (\mu_2,\nu_2))}\ \oplus\text{-R.}$$

5. Introduction of $\otimes$ in the antecedent:

$$\frac{(\mu_1,\nu_1) \leq_k (\mu,\nu), \Omega \mapsto \Gamma}{((\mu_1,\nu_1) \otimes (\mu_2,\nu_2)) \leq_k (\mu,\nu), \Omega \mapsto \Gamma}\ \otimes\text{-L.}$$

[1] We use a multi succedent reformulation of a single succedent calculus of [4,23].

6. Introduction of $\oplus$ in the consequent:

$$\frac{\Omega \mapsto \Gamma, (\mu,\nu) \leq_k (\mu_1,\nu_1);\ \ \Omega \mapsto \Gamma, (\mu,\nu) \leq_k (\mu_2,\nu_2)}{\Omega \mapsto \Gamma, (\mu,\nu) \leq_k ((\mu_1,\nu_1) \otimes (\mu_2,\nu_2))},\ \otimes\text{-R}.$$

7. Introduction of $\otimes$, $\oplus$, $\vee$ in the antecedent:

$$\frac{(\mu_1,\nu_1) \leq_k (\mu',\nu'),\ (\mu_2,\nu_2) \leq_k (\mu'',\nu''), \Omega \mapsto \Gamma}{((\mu_1,\mu_2) \odot (\mu_2,\nu_2)) \leq_k ((\mu',\nu') \odot (\mu'',\nu'')), \Omega \mapsto \Gamma}\ \odot\text{-L},$$

where $\odot$ is one of $\otimes$, $\oplus$, $\vee$.

8. Introduction of $\otimes$, $\oplus$, $\wedge$ in the consequent:

$$\frac{\Omega \mapsto \Gamma, (\mu_1,\nu_1) \leq_k (\mu',\nu'), (\mu_2,\nu_2) \leq_k (\mu'',\nu'')}{\Omega \mapsto \Gamma, ((\mu_1,\mu_2) \odot (\mu_2,\nu_2)) \leq_k ((\mu',\nu') \odot (\mu'',\nu''))}\ \odot\text{-R},$$

where $\odot$ is one of $\otimes$, $\oplus$, $\wedge$.

All the properties of bilattice operations we assumed when working with sequent calculus for BAL are assumed here too. For example, $(\frac{1}{3},\frac{2}{3}) \otimes (\frac{2}{3},\frac{1}{3})$ can be substituted by $(\frac{1}{3},\frac{1}{3})$ throughout the proof. This includes all the lattice axioms and distributivity.

Example 4. A many-sorted version of Example 2 can be proven using the sequent rules of MSL and the rule 8 above. That is, we can prove that$\forall I[\exists(\mu_1,\nu_1),(\mu_2,\nu_2)$ $(F_1 \wedge F_2 \wedge ((\mu_1,\nu_1)\varepsilon I) \wedge ((\mu_2,\nu_2)\varepsilon I) \wedge ((0,1) \leq_k ((\mu_1,\nu_1) \wedge (\mu_2,\nu_2))))]$ follows from $[\forall I \exists(\mu_1,\nu_1)(F_1 \wedge ((\mu_1,\nu_1)\varepsilon I)\wedge\ ((1,0)\ \leq_k\ (\mu_1,\nu_1))] \wedge [\forall I \exists(\mu_2,\nu_2)(F_2 \wedge ((\mu_2,\nu_2)\varepsilon I) \wedge ((0,1) \leq_k (\mu_2,\nu_2))]$.

Now we can prove that BAL and MSL* are deductively equivalent.

Theorem 2. *For any annotated formula φ the following holds:*

$$\vdash_{\mathrm{BAL}} \varphi \textit{ iff } \vdash_{\mathrm{MSL}^*} \mathrm{TRANSL}^*(\varphi).$$

Proof. The proof that $\vdash_{\mathrm{MSL}^*} \mathrm{TRANSL}^*(\varphi)$ implies $\vdash_{\mathrm{BAL}} \varphi$ is trivial. The "if" part of the proof proceeds by considering translations of axioms of BAL into proofs in MSL*, and rules for BAL into proofs in MSL*. Namely, for each rule in BAL, we translate the lower sequent of this rule into MSL*, and then show how the translation of the upper sequent(s) of the BAL rule can be derived in MSL*. For example, we take the rule DA from the sequent calculus for BAL. We fix F from the rule to be $R(\overline{x})$. The lower sequent of this rule can be translated into $\forall I \exists(\mu'',\nu'')(R(\overline{x}) \wedge I(\overline{x},(\mu'',\nu'')) \wedge (\mu',\nu') \leq_k (\mu'',\nu''), \Omega \mapsto \varphi$. We also assume that $((\mu,\nu) \leq_k (\mu',\nu'))$ is added to the right hand side of each sequent: this is needed because the condition $((\mu,\nu) \leq_k (\mu',\nu'))$ is attached to the rule DA in BAL. Being put into antecedent of a lower sequent, this translated formula will receive a derivation from the three upper sequents: $\Omega, R(\overline{x}) \mapsto \varphi$; $(\overline{x},(\mu'',\nu'')\varepsilon I), \Omega \mapsto \varphi$; and $(\mu,\nu) \leq_k (\mu'',\nu''), \Omega \mapsto \varphi$. To obtain this, one would need to apply, one after another, MSL* rules $\forall$-L, $\exists$-L, $\wedge$-L, and Tr-L. The three upper sequents will also give a proof for

$\forall I \exists (\mu'', \nu'')((R(\overline{x}) \wedge I(\overline{x}, (\mu'', \nu'')) \wedge (\mu, \nu) \leq_k (\mu'', \nu''), \Omega \mapsto \varphi$, that is, the translation of the upper sequent of the DA rule $R(\overline{x}) : (\mu, \nu), \Omega \mapsto \varphi$. We consider similarly all the rules in BAL.

We are ready to state completeness of the sequent calculus for BAL as a corollary from the Soundness and Completeness Theorem for MSL [23].

Corollary 2. *For every formula φ in BAL, if $\models_{\mathrm{BAL}} \varphi$, then $\vdash_{\mathrm{BAL}} \varphi$.*

Proof. Follows from Corollary 1, Theorem 2, and Soundness and Completeness Theorem for MSL, the latter theorem is proven in [23].

The results we have described in this section can be obtained, with minor modifications, for most of lattice and bilattice based annotated languages, such as [13,15,16,22,24,25].

The many-sorted logic MSL* we defined here is in fact just a fragment of a very general many-sorted logic MSL, [23]. It is curious that the structure of MSL* is similar to the structure of the fragment of MSL which gave a translation for a second-order logic in [23]. This shows that annotated first-order many-valued logics can be equivalently represented by conventional second-order logic with sorts.

Furthermore, the way of introducing higher-order relations I in MSL* is similar to the way how [23] introduced first-order relations when translating propositional dynamic logic into many-sorted logic. It is likely that a many-sorted representation of a multimodal logic of [2], for example, will bring into the light a close connection between annotated many-valued and multimodal logics.

5 Conclusions and Further Work

We have built a sequent calculus for a very general annotated logic BAL. We used this generality to show, using the example of BAL, that annotated many-valued logics can be syntactically, semantically, and deductively translated into conventional many-sorted logic in the style of Manzano [23]. The resulting many-sorted sequent calculus has a simpler and clearer rule representation and works within conventional many-sorted language with no semantical annotations, and hence in the future may yield some conventional structural (e.g., categorical) analysis.

The uniform framework of MSL allowed us to compare properties of BAL with other non-classical logics (second order, dynamic) which have been translated into many-sorted logics already. In the future, it may be fruitful to find a many-sorted representation of a multimodal logic of [2] and show its relations with many-valued annotated logics. This would link nicely modal and many-valued logics.

Some work has been done on practical implementation of many-sorted translation to Bilattice-based Annotated Logic Programs (BAPs), see [17]. The translation made in [17] helped to simplify certain resolution rules for BAL. A rigorous analysis of efficiency of BAPs comparing with their many-sorted analogues is to be done in the future.

The further work may include the similar analysis of other many-valued annotated logics, such as logics of [7,13,15,16,21,22,24,25], and some other lattice or bilattice based logics. This may lead to establishing a nice uniform framework for analysing different annotated lattice and bilattice based logics, their model and deductive properties.

References

1. Baaz, M., Fremuller, C.G., Sazler, G.: Automated deduction for many-valued logics. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. 2, pp. 1355–1402. Elsevier, Amsterdam (2001)
2. Baldoni, M.: Normal Multimodal Logics: Automatic Deduction and Logic Programming extension. PhD thesis, Torino, Italy (2003)
3. Church, A.: Introduction to Mathematical Logic. Princeton (1944)
4. Ebbinghaus, H.D., Flum, J., Thomas, W.: Mathematical Logic. Springer, Berlin (1984)
5. Fitting, M.: Bilattices in logic programming. In: Epstein, G. (ed.) The twentieth International Symposium on Multiple-Valued Logic, pp. 238–246. IEEE, Washington (1990)
6. Fitting, M.: Bilattices and the semantics of logic programming. Journal of logic programming 11, 91–116 (1991)
7. Fitting, M.: Many-valued modal logics. Fundamenta informaticae 15, 234–235 (1992)
8. Fitting, M.: Kleene's three-valued logics and their children. Fundamenta informaticae 20, 113–131 (1994)
9. Fitting, M.: Tableaus for many-valued modal logic. Studia Logica 55, 63–87 (1995)
10. Fitting, M.: Bilattices are nice things. Self-Reference, pp. 53–77 (2006)
11. Ginsberg, M.L.: Multivalued logics: a uniform approach to reasoning in artificial intelligence. Computational Intelligence 4, 265–316 (1988)
12. Grätzer, G.: General Lattice Theory. Birkauser Verlag, Basel, Switzerland (1978)
13. Hähnle, R.: Commodious axiomatizations of quantifiers in multiple-valued logic. Studia Logica 61(1), 101–121 (1998)
14. Hähnle, R., Escalado-Imaz, G.: Deduction in many-valued logics: a survey. Mathware and soft computing IV(2), 69–97 (1997)
15. Kifer, M., Lozinskii, E.L.: RI: A logic for reasoning with inconsistency. In: Proceedings of the 4th IEEE Symposium on Logic in Computer Science (LICS), pp. 253–262. IEEE Computer Press, Asilomar (1989)
16. Kifer, M., Subrahmanian, V.S.: Theory of generalized annotated logic programming and its applications. Journal of logic programming 12, 335–367 (1991)
17. Komendantskaya, E.: A many-sorted semantics for many-valued annotated logic programs. In: Proceedings of the Fourth Irish Conference on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT), pp. 225–229, Cork, Ireland, (August 1–5, 2006)
18. Komendantskaya, E., Power, J.: Fibrational semantics for many-valued logic programs, Submitted (2007)

19. Komendantskaya, E., Seda, A.K., Komendantsky, V.: On approximation of the semantic operators determined by bilattice-based logic programs. In: Proceedings of the Seventh International Workshop on First-Order Theorem Proving (FTP'05), pp. 112–130, Koblenz, Germany, (September 15–17, 2005)
20. Lu, J.J.: Logic programming with signs and annotations. Journal of Logic and Computation 6(6), 755–778 (1996)
21. Lu, J.J., Murray, N.V., Rosenthal, E.: A framework for automated reasoning in multiple-valued logics. Journal of Automated Reasoning 21(1), 39–67 (1998)
22. Lu, J.J., Murray, N.V., Rosenthal, E.: Deduction and search strategies for regular multiple-valued logics. Journal of Multiple-valued logic and soft computing 11, 375–406 (2005)
23. Manzano, M.: Introduction to many-sorted logic. In: Meinke, K., Tucker, J.V. (eds.) Many-Sorted logic and its Applications, pp. 3–88. John Wiley and Sons, UK (1993)
24. Salzer, G.: MUltlog 1.0: Towards an expert system for many-valued logics. In: McRobbie, M.A., Slaney, J.K. (eds.) Automated Deduction (CADE'96). LNCS, vol. 1104, pp. 226–230. Springer, Heidelberg (1996)
25. Sazler, G.: Optimal axiomatizations of finitely-valued logics. Information and Computation 162(1–2), 185–205 (2000)

Updating Reduced Implicate Tries

Neil V. Murray[1] and Erik Rosenthal[2]

[1] Department of Computer Science, State University of New York, Albany, NY 12222, USA
nvm@cs.albany.edu
[2] Department of Mathematics, University of New Haven, West Haven, CT 06516, USA
erosenthal@newhaven.edu

Abstract. The reduced implicate trie, introduced in [11], is a data structure that may be used as a target language for knowledge compilation. It has the property that, even when large, it guarantees fast response to queries. Specifically, a query can be processed in time *linear in the size of the query* regardless of the size of the compiled knowledge base.

The knowledge compilation paradigm typically assumes that the "intractable part" of the processing be done once, during compilation. This assumption could render updating the knowledge base infeasible if recompilation is required. The ability to install updates without recompilation may therefore considerably widen applicability.

In this paper, several update operations not requiring recompilation are developed. These include disjunction, substitution of truth constants, conjunction with unit clauses, reordering of variables, and conjunction with clauses.

1 Introduction

The last decade has seen a virtual explosion of applications of propositional logic. One is *knowledge representation*, and one approach to it is *knowledge compilation*, introduced by Kautz and Selman [7]. Knowledge bases can be represented as propositional theories, often as sets of clauses, and the propositional theory can then be *compiled*; i.e., preprocessed to a form that admits fast response to queries. While knowledge compilation is intractable, it is done once, in an off-line phase, with the goal of making frequent on-line queries efficient.

The answering such queries in time polynomial (indeed, often linear) in the size of the compiled theory is not very fast if the compiled theory is exponential in the size of the underlying propositional theory. As a result, many investigators have focused on minimizing the size of the compiled theory, possibly by restricting or approximating the original theory. Another approach, introduced in [11], extended in [12], and developed further in this paper, is to admit large compiled theories — stored off-line[1] — on which queries can be answered in time *linear in the size of the query*. A data structure that has this property is called a *reduced implicate trie* or, more simply, an *ri-trie* [11].

[1] The term *off-line* is used in two ways: first, for off-line memory, such as hard drives, as opposed to on-line storage, such as RAM, and secondly, for "batch preparation" of a knowledge base for on-line usage.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 183–198, 2007.

In the knowledge compilation paradigm, the emphasis is typically on performing the "intractable part" of the processing once, during compilation. In the absence of an efficient updating technology, this favors knowledge bases that are stable; i.e., a single compilation is expected to provide a repository that remains useful over a large number of queries. However, updating operations (referred to as transformations in [2]) for various target languages have been studied. Here, the first small steps are taken towards solving this problem for ri-tries.

In order that this paper be self-contained, basic notions involving formulas, clauses, implicants, and implicates are covered in the next section. The definitions of reduced implicate tries and methods for computing them are presented in Section 3. Some fairly straightforward fundamental properties of ri-tries are presented in Section 4. In Section 5, several types of updates are introduced along with techniques for achieving them. None of these updates require recompilation.

2 Preliminaries

For the sake of completeness, define an *atom* to be a propositional variable, a *literal* to be an atom or the negation of an atom, and a *clause* to be a disjunction of literals. Clauses are often referred to as sets of literals. Most authors restrict attention to *conjunctive normal form* (CNF) — a conjunction of clauses — but no such restriction is required in this paper.

Consequences expressed as minimal clauses that are implied by a formula are its *prime implicates*; (and minimal conjunctions of literals that imply a formula are its *prime implicants*). Implicates are useful in certain approaches to non-monotonic reasoning [9,14,16], where all consequences of a formula — for example, the support set for a proposed common-sense conclusion — are required. The implicants are useful in situations where satisfying models are desired, as in error analysis during hardware verification. Many algorithms have been proposed to compute the prime implicates (or implicants) of a propositional boolean formula [1,3,5,6,8,13,15,17,18].

A typical query of a propositional theory has the form, is a clause logically entailed by the theory? By definition, an *implicate* of a logical formula is a clause entailed by the formula, i.e., a clause that contains a prime implicate. So if $\mathcal{F}$ is a formula and C is a clause, then C is an implicate of $\mathcal{F}$ if (and only if) C is satisfied by every interpretation that satisfies $\mathcal{F}$. Thus asking whether a given clause is entailed by a formula is equivalent to the question, Is the clause an implicate of the formula? Throughout the paper, this question is what is meant by query.

3 Background

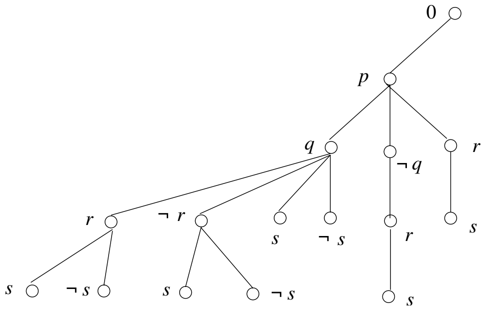
The goal of knowledge compilation is to enable fast queries. Prior approaches had the goal of a small (i.e., polynomial in the size of the initial knowledge base) compiled knowledge base. Typically, query-response time is linear, so that the efficiency of querying the compiled knowledge base depends on its size. The approach we build upon from [11] is to admit target languages that may be large as long as they enable fast queries. The idea is for the query to be processed in time *linear in the size of the query*.

Thus, if the compiled knowledge base is exponentially larger than the initial knowledge base, the query must be processed in time logarithmic in the size of the compiled knowledge base. The *ri-trie* is one data structure that admits such fast queries.

3.1 Implicate Tries

The trie is a well-known data structure introduced by Morrison in 1968 [10]; it is a tree in which each branch represents the sequence of symbols labeling the nodes[2] on that branch, in descending order. A prefix of such a sequence may be represented along the same branch by defining a special *end symbol* and assigning an extra child labeled by this symbol to the node corresponding to the last symbol of the prefix. For convenience, it is assumed here that the node itself is simply marked with the end symbol, and leaf nodes are also so marked. One common application for tries is a dictionary. The advantage is that each word in the dictionary is present precisely as a (partial) branch in the trie. Checking a string for membership in the dictionary merely requires tracing a corresponding branch in the trie. This will either fail or be done in time linear in the size of the string.

Tries have also 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. But observe that this CNF formula is significantly larger than the corresponding trie (see Figure 1). The literal that labels each interior node of the trie would appear once in each of the separate clauses represented by all branches of the subtrie rooted at that node. (The zero root in Figure 1 assures a single trie and prevents a forest.)



$$\begin{aligned}\{\ &\{p, q, r, s\},\\ &\{p, q, r, \neg s\},\\ &\{p, q, \neg r, s\},\\ &\{p, q, \neg r, \neg s\},\\ &\{p, q, s\},\\ &\{p, q, \neg s\},\\ &\{p, \neg q, r, s\},\\ &\{p, r, s\}\ \}\end{aligned}$$

Fig. 1. A Trie and its CNF Equivalent

In fact, the trie can be interpreted directly as an NNF formula, recursively defined as follows: A trie consisting of a single node represents the constant labeling 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.

[2] Many variations have been proposed in which arcs rather than nodes are labeled, and the labels are sometimes strings rather than single symbols.

Suppose first that all implicates are stored in the trie; the result is called an *implicate trie*. To define it formally, let $p_1, p_2, ..., p_n$ be the variables that appear in the input knowledge base $\mathcal{D}$, and let q_i be the literal p_i or $\neg p_i$. Literals are ordered as follows: $q_i \prec q_j$ iff $i < j$. (This can be extended to a total order by defining $\neg p_i \prec p_i, 1 \leq i \leq n$. But neither queries nor branches in the trie will contain such complementary pairs.) The implicate trie for $\mathcal{D}$ is a tree defined as follows: If $\mathcal{D}$ is a tautology (contradiction), the tree consists only of a root labeled 1 (0). Otherwise, it is a tree whose root is labeled 0 and has, for any implicate $C = \{q_{i_1}, q_{i_2}, \ldots, q_{i_m}\}$, a child labeled q_{i_1}, which is the root of a subtree containing a branch with labels corresponding to $C - \{q_{i_1}\}$. The clause C can then be checked for membership in time linear in the size of C, simply by traversing the corresponding branch.

Note that the node on this branch labeled q_{i_m} will be marked with the end symbol. Furthermore, given any node labeled by q_j and marked with the end symbol, if $j < n$, it will have as children nodes labeled q_k and $\neg q_k$, $j < k \leq n$, and these are all marked with the end symbol. This is an immediate consequence of the fact that a node marked with the end symbol represents an implicate which is a prefix (in particular, subset) of every clause obtainable by extending this implicate in all possible ways with the literals greater than q_j in the ordering.

3.2 Reduced Implicate Tries

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 are useful (even if trivial).

$$\begin{array}{llll}
\textbf{SR1}. & \mathcal{F} \longrightarrow \mathcal{F}[\mathcal{G}/\mathcal{G} \vee 0] & \mathcal{F} \longrightarrow \mathcal{F}[\mathcal{G}/\mathcal{G} \wedge 1] \\
\textbf{SR2}. & \mathcal{F} \longrightarrow \mathcal{F}[0/\mathcal{G} \wedge 0] & \mathcal{F} \longrightarrow \mathcal{F}[1/\mathcal{G} \vee 1] \\
\textbf{SR3}. & \mathcal{F} \longrightarrow \mathcal{F}[0/p \wedge \neg p] & \mathcal{F} \longrightarrow \mathcal{F}[1/p \vee \neg p]
\end{array}$$

Formally, these rules are defined for (arbitrary) formulas. However, they are also applicable to the tries in this paper: Any such trie corresponds to an NNF formula as described above. To simplify a trie, apply the applicable rules to the corresponding formula and construct the trie that corresponds to the resulting formula. (It is easy to see that each rule preserves NNF, and that the resulting formula will correspond to a unique trie.) As a result, ri-tries and formulas will be treated interchangeably. It is also convenient to regard ri-tries as 3-ary rather than n-ary trees, through the introduction of extra interior nodes labeled 0. Notation (using 4-tuples) that emphasizes the tree structure of these tries will be introduced and used when this viewpoint is convenient. Formula notation is used when it is felt that a more traditional viewpoint is helpful.

If $C = \{q_{i_1}, q_{i_2}, \ldots, q_{i_m}\}$ is an implicate of $\mathcal{F}$, it is easy to see that the node labeled q_{i_m} will become a leaf if these rules are applied repeatedly to the (formula represented by the) subtree of the implicate trie of $\mathcal{F}$ rooted at q_{i_m}. Moreover, the product of applying these rules to the entire implicate trie until no further applications of them remain will be a trie in which no internal nodes are marked with the end symbol and all leaf nodes are, rendering that symbol merely a convenient indicator for leaves. The result of this process is called a *reduced implicate trie* or simply an *ri-trie*. The branches of an ri-trie will in general correspond to a proper subset of all implicates and to a proper superset of the prime implicates.

Consider an example. Suppose that the knowledge base $\mathcal{D}$ contains the variables p, q, r, s, in that order, and suppose that $\mathcal{D}$ consists of the following clauses: $\{p, q, \neg s\}$, $\{p, q, r\}$, $\{p, r, s\}$, and $\{p, q\}$. Both the implicate trie and the ri-trie are indicated in Figure 2. The latter is simply the sub-trie obtained by making the circled node a leaf.

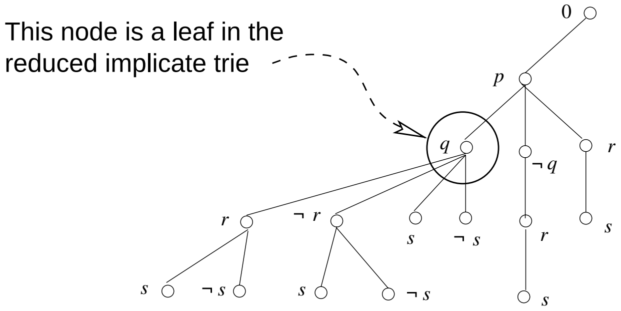


Fig. 2. Implicate Trie and ri-Trie

In the implicate trie, there are eight branches and eleven end markers representing its eleven implicates (nine in the subtree rooted at q, and one each at the two rightmost occurrences of s.)

3.3 Computing ri-Tries

The ri-trie of a formula can be obtained by applying the recursive RIT operator introduced in [11]. Let $\mathcal{F}$ be a logical formula, and let the variables of $\mathcal{F}$ be $V = \{p_1, p_2, ..., p_n\}$. Then the RIT operator is defined by

$$\text{RIT}(\mathcal{F}, V) = \begin{cases} \mathcal{F} & V = \emptyset \\ \\ \begin{array}{c} p_i \ \vee \ \text{RIT}(\mathcal{F}[0/p_i], V - \{p_i\}) \\ \wedge \\ \neg p_i \ \vee \ \text{RIT}(\mathcal{F}[1/p_i], V - \{p_i\}) \\ \wedge \\ \text{RIT}((\mathcal{F}[0/p_i] \ \vee \ \mathcal{F}[1/p_i]), V - \{p_i\}) \end{array} & p_i \in V \end{cases}$$

where p_i is the variable of lowest index in V.

Implicit in this definition is the use of simplification rules **SR1**, **SR2**, and **SR3**.

A *prefix* of a clause $\{q_1, q_2, ..., q_k\}$ is defined to be a clause of the form $\{q_1, q_2, ..., q_i\}$, where $0 \leq i \leq k$. Implicit in this definition is a fixed ordering of the variables; also, if $i = 0$, then the prefix is the empty clause.

Theorems 1 and 2 below are taken from [11]. Essentially, they guarantee that the RIT operator produces ri-tries. In other words, the resulting trie is logically equivalent to the original formula, its branches are all implicates of the formula, and any implicate has a unique prefix that corresponds precisely to a branch.

Theorem 1. If $\mathcal{F}$ is any logical formula with variable set V, then $\mathrm{RIT}(\mathcal{F}, V)$ is logically equivalent to $\mathcal{F}$, and each branch of $\mathrm{RIT}(\mathcal{F}, V)$ is an implicate of $\mathcal{F}$. □

Theorem 2. 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 prefix of C that is a branch of $\mathrm{RIT}(\mathcal{F}, V)$. □

Let $\mathrm{Imp}(\mathcal{F})$ denote the set of all implicates of $\mathcal{F}$. A pseudocode algorithm that produces ri-tries was given in [11]. The algorithm relies heavily on the following lemma:

Lemma 1. Given logical formulas $\mathcal{F}$ and $\mathcal{G}$, $\mathrm{Imp}(\mathcal{F} \vee \mathcal{G}) = \mathrm{Imp}(\mathcal{F}) \cap \mathrm{Imp}(\mathcal{G})$. □

The lemma assures us that the branches produced by the third conjunct of the RIT operator are precisely the branches that occur in both of the first two (ignoring, of course, the root labels p_i and $\neg p_i$). In particular, the recursive call $\mathrm{RIT}((\mathcal{F}[0/p_i] \vee \mathcal{F}[1/p_i]), V - \{p_i\})$ is avoided. This is significant because that call doubles the size of the formula *along a single branch*. Further use of this lemma is made in Section 5.

4 ri-Tries as Dags

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 must also preserve labels: $Label(A) = Label(f(A))$.

For the remainder of this paper, we will assume all isomorphisms to be label-preserving.

Theorem 3. Let $\mathcal{R}$ be a reduced implicate trie, and let f be an isomorphism from $\mathcal{R}$ to $\mathcal{R}$. Then f is the identity map on $\mathcal{R}$.

Proof. We proceed by induction on the number of variables in $\mathcal{R}$. The result is trivial if there is only one variable, so assume true for any ri-trie with at most n variables, and suppose $\mathcal{R}$ has $n + 1$ variables. The root has at most three children, labeled $p_1, \neg p_1, 0$. Since each has a distinct label and f is label preserving, f must map each of these children to itself. Note that the edge preservation property of f ensures that f maps each subtrie to itself. The induction hypothesis thus applies to each subtrie, and the proof is complete. □

The theorem, while straightforward, is not immediate. In Figure 3, the dag has a label-preserving isomorphism that is not the identity because it swaps the nodes labeled ”a”.

The induction of the last theorem can easily be adapted to prove

Theorem 4. Let $\mathcal{F}$ and $\mathcal{G}$ be logically equivalent formulas. Then, with respect to a fixed variable ordering, RIT($\mathcal{F}$) is isomorphic to RIT($\mathcal{G}$).

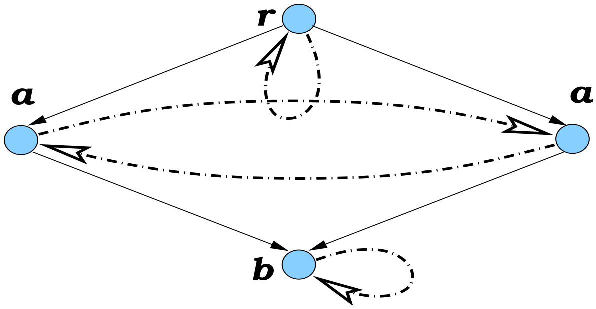


Fig. 3. Non-Identity Label-Preserving Isomorphism

Note that $\mathcal{F}$ and $\mathcal{G}$ may have different variable sets. In that case, we assume that the fixed ordering in the theorem refers to the union of these variable sets. As a result, comparing the ri-tries of two formulas amounts to a (not necessarily practical) test for logical equivalence. On the other hand, if the formulas are known to be equivalent, attention can be restricted to variables in the symmetric difference of their variable sets; all others are redundant.

5 Updating ri-Tries

It is typical in the knowledge compilation paradigm to assume that the intractable part of the processing is done only once (or perhaps just not very often). In the absence of an efficient updating technology, this favors knowledge bases that are stable; i.e., a single compilation is expected to provide a repository that remains useful over a large number of queries. The original knowledge base can always be modified and then recompiled, but in general this is expensive. As a result, updates that can be installed into the compiled knowledge base without recompiling have the potential to considerably widen applicability.

Several operations on ri-tries are developed in this section; unless stated otherwise, the variable set under consideration will be assumed to be fixed. The *intersection operation* is defined precisely and proved to be correct. This is essentially the process captured informally by the pseudocode routine called $buildzero$ from [11]. Computing the intersection of the implicate sets of two ri-tries is one way to implement their disjunction. The ri-trie produced is precisely the result of applying the RIT operator produces to the disjunction of the two original formulas (with respect to a fixed variable ordering). Thus, given a formula $\mathcal{F}$ compiled into ri-trie $\mathcal{T}_{\mathcal{F}}$, we can compute the ri-trie for $\mathcal{F} \vee \mathcal{G}$ by compiling $\mathcal{G}$ to $\mathcal{T}_{\mathcal{G}}$ and then computing the intersection of $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{G}}$.

The process of substituting truth values for variables in an ri-trie is also investigated. Even this apparently simple operation requires some care: Preserving equivalence is trivial, but preserving the prefix property is not.

A third operation that developed in this section conjoins a unit clause to an ri-trie. This is a rather specialized update that may not very useful in its own right, but it plays an important role in the operation of conjoining a clause.

The problem of reordering the variables in an ri-trie is also addressed. Suppose formula $\mathcal{F}$ has been compiled into ri-trie $\mathcal{T}_{\mathcal{F}}$. The techniques developed for unit conjunction can be employed to compute a new ri-trie for $\mathcal{F}$ (without recompiling) in which the only change is in the ordering: The i^{th} variable is made first.

Finally, it is shown how the ri-trie for a formula $\mathcal{F}$ conjoined with a clause C can be computed directly from $\mathcal{T}_{\mathcal{F}}$, the ri-trie for $\mathcal{F}$, without recompiling. This is accomplished by employing conjunction with units, variable reordering, and intersection.

A detailed analysis of the efficiency of the operations developed here is beyond the scope of this paper. Nevertheless, it is easy to see that other than conjoining a clause, all operations are no worse than linear in the size of the ri-trie. The operations could require visiting every node, but at each visited node the computation is $\mathbf{O}(1)$. Conjoining a clause is accomplished by executing a loop whose duration is proportional to the size of the clause. The practicality of these operations is likely to be ultimately decided by experimentation. But the goal of knowledge compilation is to avoid recompilation; updating is therefore a desirable alternative.

5.1 Intersecting ri-Tries

Given any 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 that represents the intersection of the implicate sets with respect to the given variable ordering. By Theorems 1 and 4, and Lemma 1, this is the ri-trie for $\mathcal{F} \vee \mathcal{G}$. This definition captures the computation expressed in pseudocode as the function $buildzero$ in [11]. There, the entire pseudocode algorithm represents the recursive operator RIT that defines the ri-trie for a formula.

Similarly, the INT operator can be defined recursively. It is again assumed for convenience that the trie is represented as a ternary tree rather than as an n-ary trie. The root of the entire trie is 0 (for non-tautologies), and any node at level i has ordered children that are either empty or are tries whose roots are labeled p_{i+1}, $\neg p_{i+1}$, and 0. As a result, a trie $\mathcal{T}$ rooted at p_i can be represented notationally as a 4-tuple $< p_i,\ \mathcal{T}^+,\ \mathcal{T}^-,\ \mathcal{T}^0 >$. This trie represents the formula $p_i \vee (\mathcal{T}^+ \wedge \mathcal{T}^- \wedge \mathcal{T}^0)$, and we write $\mathcal{T} - p_i$ to denote the second disjunct (which is equivalent to $\mathcal{T} \wedge \neg p_i$). The predicate $leaf$ returns $true$ whenever all sub-tries are empty.

Given two identically ordered tries $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{G}}$, $\text{INT}(\mathcal{T}_{\mathcal{F}}, \mathcal{T}_{\mathcal{G}})$ is defined as follows.

$$\text{INT}(\mathcal{T}_{\mathcal{F}}, \mathcal{T}_{\mathcal{G}}) \;=\; \begin{cases} \emptyset & \mathcal{T}_{\mathcal{F}} = \emptyset \vee \mathcal{T}_{\mathcal{G}} = \emptyset \\ \mathcal{T}_{\mathcal{F}} & \mathit{leaf}(\mathcal{T}_{\mathcal{G}}) \\ \mathcal{T}_{\mathcal{G}} & \mathit{leaf}(\mathcal{T}_{\mathcal{F}}) \\ \emptyset & \mathit{leaf}(\mathcal{T}_{\mathcal{F}} \bigoplus \mathcal{T}_{\mathcal{G}}) \\ \mathcal{T}_{\mathcal{F}} \bigoplus \mathcal{T}_{\mathcal{G}} & otherwise \end{cases}$$

where $\mathcal{T}_{\mathcal{F}} \bigoplus \mathcal{T}_{\mathcal{G}} \;=\; < r,\ \text{INT}(\mathcal{T}_{\mathcal{F}}^+, \mathcal{T}_{\mathcal{G}}^+),\ \text{INT}(\mathcal{T}_{\mathcal{F}}^-, \mathcal{T}_{\mathcal{G}}^-),\ \text{INT}(\mathcal{T}_{\mathcal{F}}^0, \mathcal{T}_{\mathcal{G}}^0) >$ and r is the root label of both $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{G}}$.

First note that the INT operator clearly produces a labeled trie that has the structure of a ternary tree exactly like its arguments. Lemma 2 and Theorem 5 show that this trie is precisely the ri-trie that is the intersection of its arguments. Observe also that in case four, the leaf test on $\mathcal{T}_{\mathcal{F}} \bigoplus \mathcal{T}_{\mathcal{G}}$ is required. When neither argument is a leaf (as ruled out by previous cases) and yet the intersections of all corresponding sub-tries are empty, $\mathcal{T}_{\mathcal{F}} \bigoplus \mathcal{T}_{\mathcal{G}}$ produces a leaf, but the two tries share no branches. Finally, note that when considering the implicates corresponding to a branch in an ri-trie, the zero labels are ignored.

Lemma 2. Let $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{G}}$ be ri-tries having the same variable ordering. Let $C_{\mathcal{F}}$ be a non-empty prefix of $C_{\mathcal{G}}$, where $C_{\mathcal{F}}$ is a branch in $\mathcal{T}_{\mathcal{F}}$ and $C_{\mathcal{G}}$ is a branch in $\mathcal{T}_{\mathcal{G}}$. Then $C_{\mathcal{G}}$ is a branch in $\mathrm{INT}(\mathcal{T}_{\mathcal{F}}, \mathcal{T}_{\mathcal{G}})$.

Proof. By induction on n, the number of literals in $C_{\mathcal{F}}$.

If $n = 1$, then $C_{\mathcal{F}} = \{p_i\}$ is a singleton; it is also a branch in $\mathcal{T}_{\mathcal{F}}$, which in turn must be a root leaf. So case 3 in the definition of INT applies. The intersection will be $\mathcal{T}_{\mathcal{G}}$, and $C_{\mathcal{G}}$ is by definition a branch in $\mathcal{T}_{\mathcal{G}}$ and thus also in the intersection.

Otherwise, assume true for $1 \leq n \leq k$, and suppose $n = k + 1$. Let p_i be the first literal in $C_{\mathcal{F}}$. Since both $C_{\mathcal{F}}$ and $C_{\mathcal{G}}$ correspond to branches of length greater than one, case 5 must apply. Clearly, p_i is the root label of $\mathcal{T}_{\mathcal{F}}$ and of $\mathcal{T}_{\mathcal{G}}$. Each of $C_{\mathcal{F}} - \{p_i\}$ and $C_{\mathcal{G}} - \{p_i\}$ is non-empty, and the former is a prefix of the latter.

As a result, $C_{\mathcal{F}} - \{p_i\}$ is a branch in $\mathcal{T}_{\mathcal{F}}^{+}$, $\mathcal{T}_{\mathcal{F}}^{-}$, or $\mathcal{T}_{\mathcal{F}}^{0}$; without loss of generality say $\mathcal{T}_{\mathcal{F}}^{+}$. Then $C_{\mathcal{G}} - \{p_i\}$ is a branch in $\mathcal{T}_{\mathcal{G}}^{+}$. Since $C_{\mathcal{F}} - \{p_i\}$ contains at most k literals, the induction hypothesis applies. Therefore, $C_{\mathcal{G}} - \{p_i\}$ is a branch in $\mathrm{INT}(\mathcal{T}_{\mathcal{F}}^{+}, \mathcal{T}_{\mathcal{G}}^{+})$, which implies that $C_{\mathcal{G}}$ is a branch in $\mathrm{INT}(\mathcal{T}_{\mathcal{F}}, \mathcal{T}_{\mathcal{G}})$. □

Observations

1. To apply the induction hypothesis to $\mathrm{INT}(\mathcal{T}_{\mathcal{F}}^{+}, \mathcal{T}_{\mathcal{G}}^{+})$, $\mathcal{T}_{\mathcal{F}}^{+}$ and $\mathcal{T}_{\mathcal{G}}^{+}$ must themselves be ri-tries. This is immediate: they are each the ri-trie for the clause set that they represent.
2. All non-empty branches of the intersection trie are constructed via zero or more applications of case 5, followed by one application of either case 2 or case 3, in the definition of INT. When the computation terminates from case 2(3), each such branch corresponds to the identical branch in $\mathcal{T}_{\mathcal{F}}$ ($\mathcal{T}_{\mathcal{G}}$) and to a branch in $\mathcal{T}_{\mathcal{G}}$ ($\mathcal{T}_{\mathcal{F}}$) that is a prefix of the branch in the intersection.

Theorem 5. Let $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{G}}$ be the ri-tries for $\mathcal{F}$ and $\mathcal{G}$ having the same variable ordering. Then $\mathrm{INT}(\mathcal{T}_{\mathcal{F}}, \mathcal{T}_{\mathcal{G}})$ is the ri-trie that is the intersection of $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{G}}$ and, as a result, is the ri-trie for $\mathcal{F} \vee \mathcal{G}$ with respect to the given variable ordering.

Proof. Let C be an implicate of $\mathcal{F} \vee \mathcal{G}$. By Lemma 1, C is an implicate of both $\mathcal{F}$ and $\mathcal{G}$. Then by Theorem 2, there is a unique prefix $C_{\mathcal{F}}$ of C that is a branch in $\mathcal{T}_{\mathcal{F}}$; similarly, there is a unique prefix $C_{\mathcal{G}}$ of C that is a branch in $\mathcal{T}_{\mathcal{G}}$. We must show that some unique prefix of C is a branch in the intersection.

If C is the empty clause, then both $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{G}}$ are singleton roots labeled 0, so is the intersection, and there is nothing to prove. If either $C_{\mathcal{F}}$ or $C_{\mathcal{G}}$ is empty, then one of the tries is a singleton root, the intersection is the other trie, and the result is immediate.

So assume C, $C_{\mathcal{F}}$, and $C_{\mathcal{G}}$ are not empty. One or both of $C_{\mathcal{F}}$ and $C_{\mathcal{G}}$ is a prefix of the other; without loss of generality, assume $C_{\mathcal{F}}$ is a prefix of $C_{\mathcal{G}}$. Therefore the branch corresponding to $C_{\mathcal{F}}$ in $\mathcal{T}_{\mathcal{F}}$ is a prefix of the branch in $\mathcal{T}_{\mathcal{G}}$ corresponding to $C_{\mathcal{G}}$. By Lemma 2, $C_{\mathcal{G}}$ corresponds to a branch in $\mathrm{INT}(\mathcal{T}_{\mathcal{F}}, \mathcal{T}_{\mathcal{G}})$.

No other prefix C' of C can be a branch in the intersection because by Observation 2 after Lemma 2, C' would be a branch in one of $\mathcal{T}_{\mathcal{F}}$ or $\mathcal{T}_{\mathcal{G}}$. But this would violate the unique prefix property for either $C_{\mathcal{F}}$ or $C_{\mathcal{G}}$. □

Using Theorem 5, we can give the following definition for the RIT operator. It differs from the original definition introduced in [11] in that it provides a formal basis for the computation of ri-tries as ternary trees using intersection and structure sharing, exactly as embodied by the pseudocode in that paper.

$$\mathrm{RIT}(\mathcal{F}, V) \;=\; \begin{cases} \mathcal{F} & V = \emptyset \\ (p_i \vee \mathcal{B}_1) \;\wedge\; (\neg p_i \vee \mathcal{B}_2) \;\wedge\; (0 \vee \mathcal{B}_3) & p_i \in V \end{cases}$$

where p_i is the variable of lowest index in V, and

$$\begin{array}{rrcl} & \mathcal{B}_1 & = & \mathrm{RIT}(\mathcal{F}[0/p_i], V - \{p_i\}) \\ & \mathcal{B}_2 & = & \mathrm{RIT}(\mathcal{F}[1/p_i], V - \{p_i\}) \\ \text{and} & \mathcal{B}_3 & = & \mathrm{INT}(\mathcal{B}_1,\; \mathcal{B}_2) \end{array}$$

5.2 Simplifying ri-Tries

Suppose that formula $\mathcal{F}$ has been compiled into ri-trie $\mathcal{T}_{\mathcal{F}}$, and consider the problem of computing the ri-trie $\mathcal{T}'_{\mathcal{F}}$ for $\mathcal{F}[1/p_i]$. If we just compute $\mathcal{T}_{\mathcal{F}}[1/p_i]$ and perform the obvious simplifications, we can get a trie representation of $\mathcal{F}[1/p_i]$ in the sense that the branches correspond to a CNF equivalent. This is accomplished merely by removing $\neg p$ from all branches on which it occurs, and cutting off all branches at p on which it occurs (simplifying where necessary). The resulting trie, however, is not in general an ri-trie; the prefix property may no longer hold. In Figure 4, the ri-trie on the left represents the eleven implicates from Figure 2. Read as an NNF formula, it represents

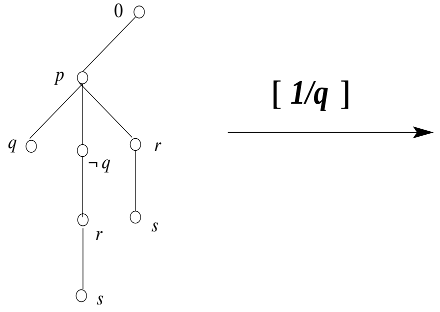


Fig. 4. Substituting a truth constant and the prefix property

$(p \vee (q \wedge (\neg q \vee r \vee s) \wedge (r \vee s)))$. The ri-trie on the right represents the result of substituting *true* (1) for q and simplifying. The resulting trie has the required prefix as a branch, but it is no longer unique. In particular, the third (rightmost) branch is unchanged but is now subsumed by the result of simplifying the second branch. It is shown below that this redundancy follows a predictable pattern and can therefore be avoided.

To deal with this problem, we define a new operation called the *prefix property preserving substitution*, written P^{3}S; it is defined as follows.

Let ri-trie $\mathcal{T} \;=\; < r,\ \mathcal{T}^+,\ \mathcal{T}^-,\ \mathcal{T}^0 >$, let the variables $V = \{p_1, \ldots, p_n\}$ of $\mathcal{T}$ be ordered by index, and let $\alpha \in \{0, 1\}$ be a truth constant. Then

$$\mathrm{P^3S}(\mathcal{T}, [\alpha/p_i], V) \;=$$

$$\begin{cases} \mathcal{T} & V = \emptyset \\ \mathcal{T}[\alpha/p_i] & i = m \wedge V = \{p_m\} \\ \begin{array}{l} < r,\ p_m \vee \mathrm{P^3S}(\mathcal{T}^+ - p_m, [\alpha/p_i], V - \{p_m\}), \\ \qquad \neg p_m \vee \mathrm{P^3S}(\mathcal{T}^- - (\neg p_m),\ [\alpha/p_i], V - \{p_m\}), \\ \qquad\qquad \mathrm{P^3S}(\mathcal{T}^0, [\alpha/p_i], V - \{p_m\}) > \end{array} & m < i \leq n \\ < r, \emptyset,\ \mathcal{T}^-[0/\neg p_m],\ \emptyset > & (i = m < n) \wedge \alpha = 1 \\ < r, \mathcal{T}^+[0/p_m],\ \emptyset,\ \emptyset > & (i = m < n) \wedge \alpha = 0 \end{cases}$$

where p_m is the variable of lowest index in V.

Theorem 6 below guarantees that P^{3}S produces exactly the right object.

Theorem 6. Let $\mathcal{T_F}$ be the ri-trie for $\mathcal{F}$ with respect to variable ordering V, where $p_i \in V$. Then $\mathrm{P^3S}(\mathcal{T_F},\ [\alpha/p_i],\ V)$ is the ri-trie for $\mathcal{F}[\alpha/p_i]$, $\alpha = 0, 1$; i.e., it is precisely the trie $\mathrm{RIT}(\mathcal{F}[\alpha/p_i], V - \{p_i\})$.

Proof. By induction on the number of applications of P^{3}S. If $n = 1$, then in the definition of P^{3}S, one of cases one, two, four, or five must apply.

Case one is that the trie is a constant, either 0 or 1; case two is that the trie simplifies to a constant after substituting for p_m, the only variable in the trie. For these cases, the lemma is immediate, so consider case four.

Note first that the roots of $\mathcal{T}^+$ and of $\mathcal{T}^-$ are p_m and $\neg p_m$, respectively. Since $\alpha = 1$, $\mathcal{T}^+$ is removed because its root has been made true; for $\mathcal{T}^-$, the root has become 0, in effect deleting $\neg p_m$ along all branches in that sub-trie. So with respect to $\mathcal{T}^+$ and $\mathcal{T}^-$, the P^{3}S operator is merely performing routine truth-functional simplifications. The crucial point is that, $\mathcal{T}^0$, which does not contain p_m, is removed.

Now consider a branch $\mathcal{B}^0 = \{p_{m+1}, \ldots, p_w\}$ of $\mathcal{T}^0$. By definition, $\mathcal{B}^0$ is in the intersection of (the sub-tries of) $\mathcal{T}^+$ and $\mathcal{T}^-$. By Lemma 2, some prefix $\mathcal{B}^- = \{p_{m+1}, \ldots, p_k\}$, $k \leq w$, of $\mathcal{B}^0$ is a branch in $\mathcal{T}^- - (\neg p_m)$. In essence removing $\neg p_m$ from any branch of $\mathcal{T}^-$ results in a branch that subsumes its corresponding branch in $\mathcal{T}^0$ (see Figure 5). Therefore, dropping $\mathcal{T}_0$ is equivalence preserving. Also, $\mathcal{T}^-[0/\neg p_m]$ must have the prefix property because otherwise $\mathcal{T}^-$ would not, contrary to hypothesis.

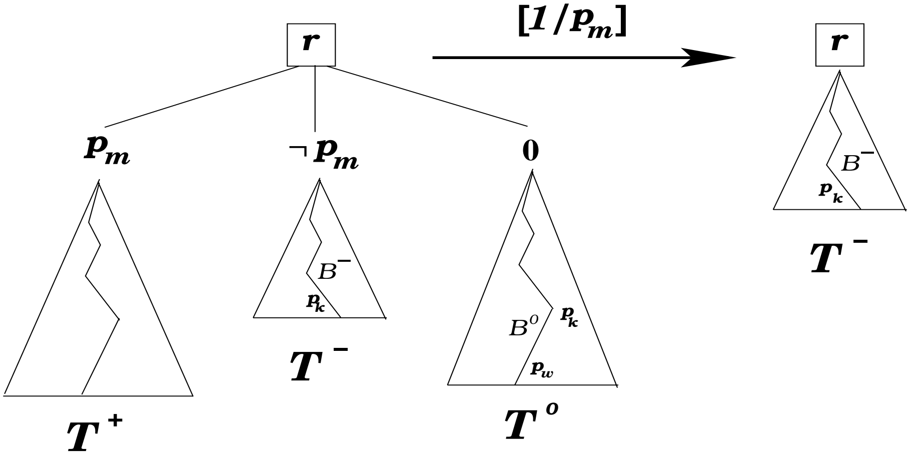


Fig. 5. Case four of $\mathrm{P^3S}$

We have shown that $\mathcal{T}^-[1/p_m]$ has the prefix property and is logically equivalent to $\mathcal{F}[1/p_m]$. But then Theorem 4 guarantees that this is identical to $\mathrm{RIT}(\mathcal{F}[1/p_m], V - \{p_m\})$. The proof for case five is exactly the dual, and the base cases are now proven.

Assume now that the theorem holds whenever $\mathrm{P^3S}$ is applied at most k times, and suppose $k+1$ applications are required to compute $\mathrm{P^3S}(\mathcal{T}, [\alpha/p_i], V)$. The first call to $\mathrm{P^3S}$ must use case three; the root is 0 because this is the initial call on the entire ri-trie, and the result is

$$\begin{aligned}< 0,\ & p_1 \vee \mathrm{P^3S}(\mathcal{T}^+ - p_1, [\alpha/p_i], V - \{p_1\}),\\ & \neg p_1 \vee \mathrm{P^3S}(\mathcal{T}^- - (\neg p_1),\ [\alpha/p_i], V - \{p_1\}),\\ & \mathrm{P^3S}(\mathcal{T}^0, [\alpha/p_i], V - \{p_1\}) > .\end{aligned}$$

The recursive calls to $\mathrm{P^3S}$ will require at most k applications; the induction hypothesis applies in each. As a result, this is precisely

$$\begin{aligned}< 0,\ & p_1 \vee \mathrm{RIT}((\mathcal{T}^+ - p_1)[\alpha/p_i], (V - \{p_1\}) - \{p_i\}),\\ & \neg p_1 \vee \mathrm{RIT}((\mathcal{T}^- - (\neg p_1))[\alpha/p_i], (V - \{p_1\}) - \{p_i\}),\\ & \mathrm{RIT}(\mathcal{T}^0[\alpha/p_i], (V - \{p_1\}) - \{p_i\}) > .\end{aligned}$$

As formulas, $\mathcal{T}^+ - p_1 = \mathcal{F}[0/p_1]$, $\mathcal{T}^- - (\neg p_1) = \mathcal{F}[1/p_1]$, and $\mathcal{T}^0 = (\mathcal{F}[0/p_1] \vee \mathcal{F}[1/p_1])$. The formulas may be substituted for the tries and by Theorem 4, the second, third, and fourth elements of the tuple are unchanged; the result is

$$\begin{aligned}< 0,\ & p_1 \vee \mathrm{RIT}(\mathcal{F}[0/p_1][\alpha/p_i], (V - \{p_i\}) - \{p_1\}),\\ & \neg p_1 \vee \mathrm{RIT}(\mathcal{F}[1/p_1][\alpha/p_i], (V - \{p_i\}) - \{p_1\}),\\ & \mathrm{RIT}((\mathcal{F}[0/p_1][\alpha/p_i] \vee \mathcal{F}[1/p_1][\alpha/p_i]), (V - \{p_i\}) - \{p_1\}) > .\end{aligned}$$

The boolean replacements can be commuted, the root is 0, and so the trie represents the conjunction

$$\begin{array}{c} p_1 \vee \mathrm{RIT}(\mathcal{F}[\alpha/p_i][0/p_1], (V - \{p_i\}) - \{p_1\}) \\ \wedge \\ \neg p_1 \vee \mathrm{RIT}(\mathcal{F}[\alpha/p_i][1/p_1], (V - \{p_i\}) - \{p_1\}) \\ \wedge \\ \mathrm{RIT}((\mathcal{F}[\alpha/p_i][0/p_1] \vee \mathcal{F}[\alpha/p_i][1/p_1]), (V - \{p_i\}) - \{p_1\}) \end{array}$$

This is precisely what results from the first expansion of $\mathrm{RIT}(\mathcal{F}[\alpha/p_i], (V - \{p_i\}))$, and this completes the proof. □

5.3 Conjoining Unit Clauses

Suppose a formula $\mathcal{F}$ has been compiled into the ri-trie $\mathcal{T}_\mathcal{F}$, and consider the problem of computing the ri-trie for $\mathcal{F} \wedge \{p\}$. (We will treat p as a variable; the dual case when it is a literal that negates its variable is straightforward.) Let $V = \{p_1, \ldots, p_n\}$ be the variables of $\mathcal{T}_\mathcal{F}$, where $p = p_i$. Suppose the RIT operator is applied to $\mathcal{T}_\mathcal{F} \wedge \{p\}$ using the variable order $V' = \{p, p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n\}$. Then

$$\mathrm{RIT}((\mathcal{T}_\mathcal{F} \wedge \{p\}), V') \;=\; \left(\begin{array}{c} p \;\vee\; \mathrm{RIT}((\mathcal{T}_\mathcal{F} \wedge \{p\})[0/p], V' - \{p\}) \\ \wedge \\ \neg p \;\vee\; \mathrm{RIT}((\mathcal{T}_\mathcal{F} \wedge \{p\})[1/p], V' - \{p\}) \\ \wedge \\ 0 \;\vee\; \mathrm{INT} \begin{pmatrix} \mathrm{RIT}((\mathcal{T}_\mathcal{F} \wedge \{p\})[0/p],\; V' - \{p\}), \\ \mathrm{RIT}((\mathcal{T}_\mathcal{F} \wedge \{p\})[1/p],\; V' - \{p\}) \end{pmatrix} \end{array} \right)$$

Since $(\mathcal{T}_\mathcal{F} \wedge \{p\})[0/p] \;=\; 0$ and $(\mathcal{T}_\mathcal{F} \wedge \{p\})[1/p] \;=\; \mathcal{T}_\mathcal{F}[1/p]$, we have

$$\mathrm{RIT}((\mathcal{T}_\mathcal{F} \wedge \{p\}), V') \;=\; \left(\begin{array}{c} p \;\vee\; \mathrm{RIT}(0,\; V' - \{p\}) \\ \wedge \\ \neg p \;\vee\; \mathrm{RIT}(\mathcal{T}_\mathcal{F}[1/p],\; V' - \{p\}) \\ \wedge \\ 0 \;\vee\; \mathrm{INT}(0,\; \mathrm{RIT}(\mathcal{T}_\mathcal{F}[1/p],\; V' - \{p\})) \end{array} \right)$$

Clearly, $\mathrm{RIT}(0,\; V' - \{p\}) = 0$ and $\mathrm{INT}(0,\; \mathrm{RIT}(\mathcal{T}_\mathcal{F}[1/p],\; V' - \{p\})) = \mathrm{RIT}(\mathcal{T}_\mathcal{F}[1/p],\; V' - \{p\})$. By Theorem 6, $\mathrm{RIT}(\mathcal{T}_\mathcal{F}[1/p],\; V' - \{p\}) = P^3S(\mathcal{T}_\mathcal{F}[1/p],\; V')$. So we now have

$$\mathrm{RIT}((\mathcal{T}_\mathcal{F} \wedge \{p\}), V') \;=\; \left(\begin{array}{c} p \\ \wedge \\ \neg p \;\vee\; P^3S(\mathcal{T}_\mathcal{F}, [1/p],\; V') \\ \wedge \\ 0 \;\vee\; P^3S(\mathcal{T}_\mathcal{F}, [1/p],\; V')) \end{array} \right)$$

and the result is the ri-trie for $\mathcal{F} \wedge \{p\}$ with respect to the new variable ordering V'. This is shown in Figure 6.

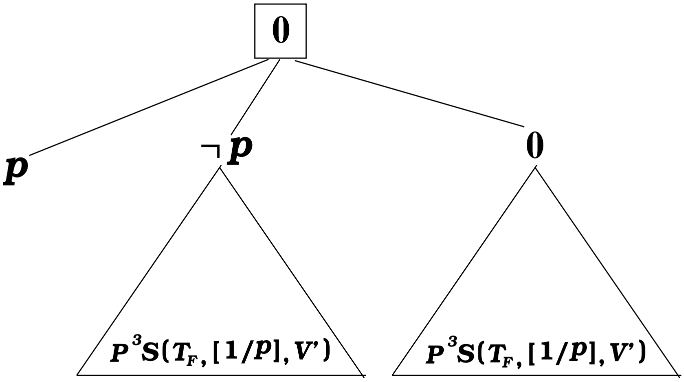


Fig. 6. Conjoining the Unit Clause $\{p\}$

The process of computing the ri-trie for $\mathcal{F} \wedge \{p\}$, given the ri-trie $\mathcal{T}_\mathcal{F}$ can now be summarized. First, compute the ri-trie for $\mathcal{F}[1/p]$ by computing $\mathcal{T}_2 = P^3S(\mathcal{F}, [1/p], V')$. Construct $\mathcal{T}_1$ from a copy of $\mathcal{T}_2$, but with the root replaced by $\neg p$. Finally, build the trie in which the root is labeled 0 and the children are leaves labeled p, $\mathcal{T}_1$, and $\mathcal{T}_2$, in that order.

5.4 Reordering ri-Tries

In the previous section, the conjunction of a unit clause was aided considerably by adopting a new variable ordering. Using a similar analysis, such a reordering (involving only one variable) can be accomplished alone, without any modification of the knowledge base represented.

Suppose $\mathcal{T}_\mathcal{F} = \mathrm{RIT}(\mathcal{F}, V)$, where $V = \{p_1, \ldots, p_n\}$. Consider the problem of computing a new ri-trie $\mathcal{T}'_\mathcal{F} = \mathrm{RIT}(\mathcal{F}, V')$, where $V' = \{p_i, p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n\}$. Once again apply the RIT operator, but this time to $\mathcal{T}_\mathcal{F}$ alone using the new variable order.

$$\mathrm{RIT}(\mathcal{T}_\mathcal{F}, V') \;=\; \left(\begin{array}{c} p_i \;\vee\; \mathrm{RIT}(\mathcal{T}_\mathcal{F}[0/p_i], V' - \{p_i\}) \\ \wedge \\ \neg p_i \;\vee\; \mathrm{RIT}(\mathcal{T}_\mathcal{F}[1/p_i], V' - \{p_i\}) \\ \wedge \\ 0 \;\vee\; \mathrm{INT} \begin{pmatrix} \mathrm{RIT}(\mathcal{T}_\mathcal{F}[0/p_i],\ V' - \{p_i\}), \\ \mathrm{RIT}(\mathcal{T}_\mathcal{F}[1/p_i],\ V' - \{p_i\}) \end{pmatrix} \end{array} \right)$$

Once again the P^3S operator and Theorem 6 prove useful. All four invocations of RIT on the right side of the above equation can be replaced by equivalent expressions that use P^3S. Let $\mathcal{T}_0 = P^3S(\mathcal{T}_\mathcal{F}, [0/p_i], V')$, $\mathcal{T}_1 = P^3S(\mathcal{T}_\mathcal{F}, [1/p_i], V')$, and $\mathcal{T}_2 = \mathrm{INT}(\mathcal{T}_0, \mathcal{T}_1)$. Then

$$\mathrm{RIT}(\mathcal{T}_\mathcal{F}, V') \;=\; (p_i \;\vee\; \mathcal{T}_0) \;\wedge\; (\neg p_i \;\vee\; \mathcal{T}_1) \;\wedge\; \mathcal{T}_2$$

5.5 Adding New Clauses

The results of the previous sections can be combined to provide a technique for updating an ri-trie to account for a new clause being added to the knowledge base. Assume that formula $\mathcal{F}$ has been compiled into ri-trie $\mathcal{T}_{\mathcal{F}}$. Now we consider the problem of computing the ri-trie $\mathcal{T}_{\mathcal{F}}^{C}$ for $\mathcal{F} \wedge C$, where $C = \{l_1, \ldots, l_m\}$.

First observe that $\mathcal{F} \wedge C \;\; = \;\; \bigvee_{i=1}^{m} (\mathcal{F} \wedge \{l_i\})$. Therefore, the ri-trie for $\mathcal{F} \wedge C$ can be computed as the intersection of the ri-tries for $(\mathcal{F} \wedge \{l_i\}), 1 \leq i \leq m$. It will be more convenient to express this as a series of pairwise intersections between $(\mathcal{T}_{\mathcal{F}} \wedge (\vee_{j=1}^{i} l_j))$ and $(\mathcal{T}_{\mathcal{F}} \wedge \{l_{i+1}\}), 1 \leq i < m$. Each unit conjunction can be computed using the techniques of Section 5.3. However, in each case, the ri-trie would be constructed with respect to a variable ordering in which the variable of the unit is first. This can be overcome with the variable reordering results from Section 5.4. This leads to the following process.

```
$\mathcal{T} \leftarrow (\mathcal{T}_{\mathcal{F}} \wedge \{l_1\})$
$i \leftarrow 1$
while $(i < m)$
    $\mathcal{T}' \leftarrow (\mathcal{T}_{\mathcal{F}} \wedge \{l_{i+1}\})$
    Recompute $\mathcal{T}$ making the variable of $l_{i+1}$ first.
    $\mathcal{T} \leftarrow \mathrm{INT}(\mathcal{T},\ \mathcal{T}')$
    $i \leftarrow i + 1$
end while
$\mathcal{T}_{\mathcal{F}}^{C} \ \leftarrow \ \mathcal{T}$
```

After termination, $\mathcal{T}_{\mathcal{F}}^{C}$ is the ri-trie for $\mathcal{F} \wedge C$.

6 Future Work

The results in this paper provide some progress in the update and maintenance of ri-tries, but many other questions remain. For example, how can the presence of new variables in clauses or formulas being conjoined to an ri-trie be handled? Are the techniques introduced here sufficient? What if information is to be removed or somehow changed? What operations are there, and which can be performed without recompiling?

There is inherent redundancy in representing implicates that lack many variables early in the ordering. Some techniques for reducing such redundancy were developed in [12], but they apply directly only to conjunctions of two formulas. Can they be extended to n-ary conjunctions? Can other reduction techniques be developed?

Acknowledgments

The authors are grateful to the referees, each of whom provided useful suggestions and constructive criticism.

References

1. Coudert, O., Madre, J.: Implicit and incremental computation of primes and essential implicant primes of boolean functions. In: Proceedings of the 29th ACM/IEEE Design Automation Conference, 36–39 (1992)
2. Darwiche, A., Marquis, P.: A knowledge compilation map. Journal of Artificial Intelligence Research 17, 229–264 (2002)
3. de Kleer, J.: An improved incremental algorithm for computing prime implicants. Proceedings of AAAI-92, San Jose, CA, pp. 780–785 (1992)
4. 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 (LNAI), vol. 3488, Springer, Heidelberg (2005)
5. Jackson, P., Pais, J.: Computing prime implicants. In: Stickel, M.E. (ed.) 10th International Conference on Automated Deduction. LNCS, vol. 449, Springer, Heidelberg (1990)
6. Jackson, P.: Computing prime implicants incrementally. In: Kapur, D. (ed.) Automated Deduction - CADE-11. LNCS, vol. 607, Springer, Heidelberg (1992)
7. Kautz, H., Selman, B.A.: general framework for knowledge compilation, in Proceedings of the International Workshop on Processing Declarative Knowledge (PDK), Kaiserslautern, Germany (July 1991)
8. Kean, A., Tsiknis, G.: An incremental method for generating prime implicants/implicates. Journal of Symbolic Computation 9, 185–206 (1990)
9. Kean, A., Tsiknis, G.: Assumption based reasoning and clause management systems. Computational Intelligence 8(1), 1–24 (1992)
10. Morrison, D.R.: PATRICIA — practical algorithm to retrieve information coded in alphanumeric. Journal of the ACM 15(4), 34–514 (1968)
11. Murray, N.V., Rosenthal, E.: Efficient query processing with compiled knowledge bases. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS (LNAI), vol. 3702, Springer, Heidelberg (2005)
12. Murray, N.V., Rosenthal, E.: Efficient Query Processing with Reduced Implicate Tries. To appear, Journal of Automated Reasoning
13. Ngair, T.A.: new algorithm for incremental prime implicate generation. Proc of IJCAI-93, 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. In: Journal of Automated Reasoning, vol. 18(3), pp. 337–356. Kluwer, Dordrecht (1997)
16. Reiter, R., de Kleer, J.: Foundations of assumption-based truth maintenance systems: preliminary report. Proceedings of the 6th National Conference on Artificial Intelligence, Seattle, WA , pp. 183-188 (July 12-17 1987)
17. Slagle, J.R., Chang, C.L., Lee, R.C.T.: A new algorithm for generating prime implicants. IEEE transactions on Computers 19(4), 304–310 (1970)
18. Strzemecki, T.: Polynomial-time algorithm for generation of prime implicants. Journal of Complexity 8, 37–63 (1992)

A Bottom-Up Approach to Clausal Tableaux

Nicolas Peltier

LIG, CNRS
46, avenue Félix Viallet
38031 Grenoble Cedex, France
Nicolas.Peltier@imag.fr

Abstract. We present a new proof procedure for first-order logic. It is close in spirit to the usual tableaux-based procedures, but uses a more compact representation of the search space. Roughly speaking, it constructs the tableau from the leaves to the root, and tries to factorize common subtrees when possible.

We study the complexity of our procedure for several propositional classes and we show that it is polynomial for all these classes.

1 Introduction

The search for efficient proof calculi is at the heart of automated deduction. Resolution-based calculi still play a prominent role (for first-order logic), but many efficient proof procedures are based on semantic tableaux. In this paper, we restrict ourselves to clausal tableaux, which allow for a number of efficiency improvements. Roughly speaking, clausal tableaux are based on the following principle: a tree labeled by literals is constructed by applying repeatedly the following (unique) expansion rule: (i) **Choose** a clause $l_1 \vee \ldots \vee l_n$ occurring in the considered clause set. (ii) **Add** n successors labeled respectively by $l_1, \ldots, l_n$. A branch is *closed* if it contains two contradictory literals. The tableau is *closed* when all branches are closed, which entails that the clause set at hand is unsatisfiable. This procedure extends easily to the first-order case, by using unification to find the instances allowing to close the branches.

As well known, an additional restriction can be imposed on the above rule, namely there must exist a *connection* [5,12] between the chosen node and the clause that is used to expand the tableau. More precisely, the clause must contain a literal l_i that is complementary to the last literal in the branch (this is called *strong connectedness*). Thus one of the branches corresponding to $l_1, \ldots, l_n$ can be immediately closed. This makes the proof procedure *non confluent*, hence backtracking must be used to find the closed tableau. However, backtracking is needed anyway in the first-order case, because it is extremely difficult to define confluent proof strategies for free variable tableaux [3,10,2,14].

Many improvements can be added to this basic procedure, in order to prune the search space [11]. Numerous refinements have been proposed, for instance the Hyper-Tableaux [1], PUHR-Tableaux [8], Model Elimination, Disconnection method [6,13] etc.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 199–215, 2007.

In this paper, we present a proof procedure for first-order logic which is based on similar ideas, but which aims at reducing the search space by "factorizing" common subspaces when possible.

The basic idea is to construct the closed tableau *bottom-up*. First, we try to identify the literals possibly labeling the *leaves* of the (closed) tableau. They can be "easily" identified because they correspond to nodes on which the closure rule can be applied. Thus it suffices either to find a unit clause or to find a "path" in the tree from a node labeled by a literal l to a node labeled by a literal l^c complementary to l[1]. This can be done relatively efficiently, because we do not need to keep track of the whole path connecting the nodes: only the *existence* of such a path is important (if propositional clause sets are considered then only $o(n^2)$ potential paths need to be considered, where n denotes the number of variables). Thus *we do not need* to construct the tableau explicitly. When the leaves are known, one can check that the tableau is closed using a bottom-up procedure. We start at the leaves and we try to reconstruct a closed tableau by propagation, applying the expansion rule "in reverse". We stop when the root can be reached.

As we shall see, this approach can be extended to the first-order case. Its main advantage is that numerous subparts of the tableau can be *shared*, because when the tableau is reconstructed bottom-up, the literals that are used to close the branches are known, which is obviously not the case in the usual procedure. As we shall see, this may yield a more compact representation of the search space. Moreover, some refinements can be added to this basic algorithm:

- First, we may restrict ourselves to leaves labeled by a literal l such that there exists a path from l to l^c *and* a path from l^c to l. Indeed, if there is no path from l^c to l then we can close a tableau starting at l^c without applying the closure rule on the literals l, l^c.
- Second, we can fix the ordering $\prec$ on which the literals are considered in a given clause. Given a clause $a \vee b$ for instance (with $a \succ b$) it is useless to explore the paths through b if the branch corresponding to a cannot be closed.
- Third, when looking for a path from l to l^c we can skip the paths traversing a literal $k \succ l$. Indeed, as we shall see, the literals can be considered and eliminated in decreasing order (according to $\succ$), which implies that k has already been eliminated when l is considered. For instance, given the clauses $\{a \vee b, a \vee \neg b, \neg a \vee b, \neg a \vee \neg b\}$ (where $a \succ b$) we can generate the path $a \rightarrow \neg a$ and $\neg a \rightarrow a$ but *not* $b \rightarrow \neg b$ and $\neg b \rightarrow b$. Thus only a and $\neg a$ are available as potential leaves.

These improvements are very important for pruning the search space, because they strongly restrict the number of paths and nodes identified as potential leaves. The formal definition of our approach will be given later, but we now

[1] A *path* through a set of clauses S is a sequence of literals $p_1, \ldots, p_n$, such that for all $i \in [1..n-1]$, there exists a clause in S containing p_i and p_{i+1}^c.

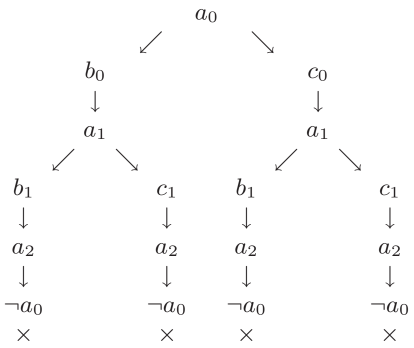


Fig. 1. A tableau proof for S_2

provide a simple example of application in order to help the reader to grasp the intuitive idea. For the sake of clarity, this example is purely propositional.

Example 1. We consider the following set of clauses $S_n = \{a_0, \neg a_n \vee \neg a_0\} \cup \{\neg a_i \vee b_i \vee c_i, \neg b_i \vee a_{i+1}, \neg c_i \vee a_{i+1} \mid i \in [0..n-1]\}$.

Starting from a_0 we obtain two different branches labeled by b_0 and c_0. Both branches are expanded by a unique node labeled by a_1, yielding again two distinct branches labeled by b_1 and c_1 respectively. This process is iterated until a_n is obtained. At this point, one can use the last clause to derive $\neg a_0$ which closes the branch. The size of the obtained tableau is $o(2^n)$ (see Figure 1)[2].

Our procedure can compute (a representation of) the *same* closed tableau in polynomial time. We assume that $a_i \succ b_i \succ c_i \succ a_{i+1}$ for any $i \in [0..n-1]$. We first try to find a path from a literal l to its complementary l^c. The only possible path is from a_0 to $\neg a_0$ (or from $\neg a_0$ to a_0). If we choose a_0 as a leaf no tableau can be constructed, since there is no non unit clause containing a_0 (thus no expansion rule can derive a_0, except if a_0 and $\neg a_0$ both occur in the clause set which is not the case). Thus we choose $\neg a_0$. Then we try to reconstruct a closed tableau. The only expansion rule that can derive $\neg a_0$ yields the predecessor node a_n. At this point, we can apply two possible expansion rules: one with the clause $\neg b_{n-1} \vee a_n$, and the other with $\neg c_{n-1} \vee a_n$. This yields two distinct branches: b_{n-1} and c_{n-1}. Again, both branches can be reduced to a_{n-1}. Obviously these two branches can be merged. Indeed, they are labeled by the same literal a_{n-1} *and* the same set of literals (namely $\{a_0\}$) is used to closed each branch. This process is iterated until a_0 is reached (see Figure 2).

We call the procedure BTC, for **B**ottom-up **T**ableaux **C**onstruction.

The rest of the paper is organized as follows. In Section 2 we introduce the basic definitions that are needed for the understanding of our work. In Section 3

[2] Of course, in this particular case, a much shorter tableau (of size $o(n)$) can be obtained by starting from the clause $a_n \vee a_0$. This simple example is chosen only to show how our procedure works and not to demonstrate its theoretical power.

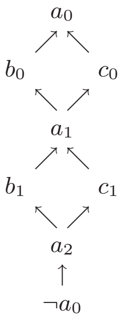


Fig. 2. Our procedure (constructing the same closed tableau for S_2)

we present a formal description of BTC. In Section 4 we establish its soundness and refutational completeness. In Section 5 we study its complexity on some particular classes of propositional formulae. We show that it runs in polynomial time for many classes, including (but not restricted to) the Horn class or the Krom class. Section 6 contains concluding remarks and some lines of future work.

2 Preliminary

The set of *terms* is built as usual on a given set of *function* and *constant* symbols and a set of *variables*. An *atom* is of the form $p(t_1, \ldots, t_n)$ where p is a *predicate symbol* and $t_1, \ldots, t_n$ are terms. A *literal* is either an atom (positive literal) or the negation of an atom. If l is a literal, then l^c denotes the complement of l, i.e. $p(\boldsymbol{t})^c \overset{\text{def}}{=} \neg p(\boldsymbol{t})$ and $(\neg p(\boldsymbol{t}))^c \overset{\text{def}}{=} p(\boldsymbol{t})$. A *clause* is a finite multiset of literals, denoted as a disjunction.

A *substitution* σ is a function from variables to terms. As usual, σ can be extended into an homomorphism on terms, atoms, literals and clauses.

A term, atom, literal, or clause is said to be *ground* if it contains no variable. A substitution is said to be *ground* if $x\sigma$ is ground, for all variables x in the domain of σ. If σ, θ are two substitutions then $\sigma\theta$ denotes the composition of σ and θ. A substitution σ is said to be *more general* than a substitution θ if there exists a substitution γ such that $\theta = \sigma\gamma$. A substitution σ is said to be a *unifier* of two terms t, s if $t\sigma = s\sigma$. Two terms, atoms or literals having a unifier are said to be *unifiable*. As well known, any unifiable pair of terms has a most general unifier (mgu), unique up to a renaming of variables.

An *interpretation* is a set of ground atoms. We write $I \models S$ (or I is a model of S) if I validates S. If l is an atom, then we have $I \models l$ iff $l \in I$ and $I \models \neg l$ iff $l \notin I$. If C is a clause, then $I \models C$ iff for all ground substitutions σ there exists $l \in C$ such that $I \models l\sigma$. If S is a set of clauses, then $I \models S$ iff $\forall C \in S, I \models C$. A set of clauses S having no model is said to be *unsatisfiable*.

3 Bottom-Up Tableaux Construction

3.1 *BTC*-formulae

Our proof procedure will be described by a set of *inference rules*, operating on a particular kind of formulae, called *BTC*-formulae. Formally, a *BTC*-formula is:

- Either of the form $(l_1 \rightarrow l_2)$ where l_1, l_2 are literals. Intuitively, $(l_1 \rightarrow l_2)$ states that there exists a path from l_1 to l_2 in the tableau.
- Or of the form $[l \mid \Phi]$ where l is a literal and Φ is a set of literals (called the *constraint*). This means that one can construct a closed tableau with root l^c, **provided** the current branch contains all the literals in Φ (i.e. for any ground substitution σ, $l\sigma$ is a logical consequence of $\Phi\sigma$).
- Or of the form $\langle C \rangle$ where C is a clause (in the usual sense). We write $\langle C \rangle$ instead of C to avoid confusions. The initial set of *BTC*-formulae contains only the formulae of the form $\langle C \rangle$ where C occurs in the considered clause set.

We extend the application of substitutions to *BTC*-formulae in a straightforward way: $(l_1 \rightarrow l_2)\sigma \overset{\text{def}}{=} (l_1\sigma \rightarrow l_2\sigma)$, $[l \mid \Phi]\sigma \overset{\text{def}}{=} [l\sigma \mid \{k\sigma \mid k \in \Phi\}]$, and $\langle C \rangle\sigma \overset{\text{def}}{=} \langle C\sigma \rangle$.

We assume there is an ordering $\succ$ among atoms, such that for any substitution σ, we have $l \succ m \Rightarrow l\sigma \succ m\sigma$. $\succ$ is extended to literals by ignoring the sign. We also assume that we are given a set Γ of literals such that for any ground literal l, either $l \in \Gamma$ or $l^c \in \Gamma$, and for any non ground literal l, if there exists a ground substitution σ such that $l\sigma \in \Gamma$, then $l \in \Gamma$. Informally, $\succ$ and Γ are used to specify the strategy.

If l, k are literals and S is a set of clauses, we write $l \vartriangle_S k$ iff either $l = k$ or if l and k^c are not unifiable and if for any (renaming of a) clause $C \in S$ containing a literal l' such that l' and l have a m.g.u. θ, $k^c\theta \in C\theta$. A clause is said to be *blocked* in S if it contains two literals l, k such that there exists m with $l^c \vartriangle_S m$ and $k^c \vartriangle_S m^c$. In particular, any tautological clause is blocked. Note that checking whether a clause is blocked can be done in polynomial time w.r.t. the size of the clause set.

Note that if $l \vartriangle_S k$ then for any ground substitution θ, we have $l\theta \vartriangle_{S'} k\theta$ where S' denotes the set of ground instances of the clauses in S. Thus, if C is blocked in S, then any ground instance of C is blocked in S'.

We extend the notion of models and satisfiability to *BTC*-formulae as follows. An interpretation I *validates a ground BTC-formula* π (written $I \models \pi$) if either π is of the form $\langle C \rangle$ and $I \models C$, or π is of the form $(l \rightarrow k)$ or π is of the form $[l \mid \Phi]$ and either $I \models l$ or there exists $l' \in \Phi$ such that $I \not\models l'$. If π is non ground, then $I \models \pi$ iff for all ground substitutions σ, $I \models \pi\sigma$.

3.2 Rules

As for the resolution calculus we assume that the variables that are shared by some of the premisses are renamed prior to any rule application. The first two rules allow one to infer *BTC*-formulae of the form $(l \rightarrow k)$.

Link

$$\frac{\langle l \vee k^c \vee C \vee l_1 \vee \ldots \vee l_n \rangle \ [l'_1 \mid \Phi_1] \ldots [l'_n \mid \Phi_n]}{(l \rightarrow k)\sigma}$$

where:

1. σ is the m.g.u of $(l_1^c, \ldots, l_n^c)$ and $(l'_1, \ldots, l'_n)$.
2. For any literal l' occurring in C, $l'\sigma \not\succ k\sigma$.
3. There is no literal in $(\Phi_1 \cup \ldots \cup \Phi_n)\sigma$ that occurs in $(l \vee k^c \vee C \vee l_1 \vee \ldots \vee l_n)\sigma$.
4. $(l \vee k^c \vee C \vee l_1 \vee \ldots \vee l_n)\sigma$ is not blocked.

Example 2. We give an example of application, explaining the meaning of the conditions 2 and 3. Assume that the clause set at hand contains a clause $a \vee b \vee c$, where $a \succ b \succ c$.

We start exploring the paths through the above clause. We consider in priority the greatest literals. The rule Link infers the following *BTC*-formulae: $(a \rightarrow \neg b), (b \rightarrow \neg a), (c \rightarrow \neg a)$. However, due to the second condition, it does **not** infer $(a \rightarrow \neg c)$ for instance, since in this case we would have $C = b$ and $b \succ c$. Intuitively, it is not necessary to explore the branch corresponding to c if no closed subtableau can be found for the branch corresponding to b.

Now, assume that $[\neg b \mid \emptyset]$ has been derived. This means that we have succeeded to find a closed subtableau of root b. This offers additional possibilities: we can derive, for instance: $(a \rightarrow \neg c)$ (we apply the rule Link with $l = a$, $k = \neg c$, $C = \emptyset$, $n = 1$ and $l_1 = b$).

Now assume that a branch $[\neg a \mid \{c\}]$ is derived. This means that there exists a closed subtableau with root a, but *only in the branches containing* c. Can we derive $(b \rightarrow \neg c)$? No because if c occurs before the root b in the tableau, then the clause $a \vee b \vee c$ holds in the corresponding branch, hence does not need to be considered. Thus there is no need to explore the paths through it (this justifies Condition 3, which guarantees that the constructed tableau is *regular*).

Transitivity

$$\frac{(l_1 \rightarrow l_2), (l'_2 \rightarrow l_3)}{(l_1 \rightarrow l_3)\sigma}$$

where $l_2\sigma \not\succ l_3\sigma$, σ is the mgu of l_2, l'_2.

This rule is very natural: it simply appends two paths. Notice that the condition $l_2\sigma \not\succ l_3\sigma$ imposes strong limitation on the applicability of the rule. It states than when searching for a path from l to k, one does not have to consider the paths through the literals that are strictly greater than k.

Example 3. Let $S = \{p(x) \vee q(x,a), \neg q(b,y) \vee p(y)\}$.

Assume to simplify that $\succ$ is empty. The rule Link derives $(p(x) \rightarrow \neg q(x,a))$ and $(\neg q(b,y) \rightarrow \neg p(y))$. Then the Transitivity rule derives $(p(b) \rightarrow \neg p(a))$.

The rule below detects literals that can be added in the constraint part of the *BTC*-formula. If there exists a (non-trivial) path from a literal l to l^c ($l^c = k$ in the rule above) and a path from l^c (i.e. k) to l, then l (or l^c) is eligible. The actual sign of the literal (l or l^c) is chosen according to the set Γ. The rule uses unification to find relevant instances, thus additional first-order literals k, l', k' need to be considered, where k and l' are unified with l^c and k' with l.

Leaf Detection

$$\frac{(l \to k),\ (l' \to k')}{[l \mid l]\sigma}$$

If σ is a m.g.u of l, k^c, l'^c, k', and $l \in \Gamma$.

This rule allows to identify potential leaves, as explained in the Introduction. $[l \mid l]$ states that a closed tableau for l^c can be constructed in branches containing l, which is of course trivial ($l \Rightarrow l$).

Example 4. Let $S = \{p(x,y) \vee q(x), \neg q(a) \vee p(a,b), \neg p(x,y) \vee q(x), \neg q(x) \vee \neg p(x,z)\}$. We derive using Link the *BTC*-formulae: $(p(x,y) \to \neg q(x))$, $(\neg q(a) \to \neg p(a,b))$, $(\neg p(x,y) \to \neg q(x))$, $(\neg q(x) \to p(x,z))$. Then by Transitivity we get: $(p(a,y) \to \neg p(a,b))$ and $(\neg p(x,y) \to p(x,z))$. Finally, the Leaf Detection rule applies yielding: $[p(a,b) \mid \{p(a,b)\}]$ or $[\neg p(a,b) \mid \{\neg p(a,b)\}]$.

The following three rules reconstruct the tableau from the leaves to the root.

Propagation

$$\frac{\langle l_1 \vee \ldots \vee l_n \vee l\rangle, [l'_1 \mid \Phi_1], \ldots, [l'_n \mid \Phi_n]}{[l \mid \Phi_1 \cup \ldots \cup \Phi_n]\sigma}$$

where σ is the mgu of $(l_1^c, \ldots, l_n^c)$ and $(l'_1, \ldots, l'_n)$.

Note that we may have $n = 0$ (in this case a formula $[l \mid \emptyset]$ is derived from $\langle l\rangle$). If $\Phi_1 = \ldots = \Phi_n = \emptyset$ then the Propagation rule is essentially equivalent to unit resolution. Otherwise, the Propagation rule is a form of resolution inference, but in which one is only allowed to resolve on one particular literal in the clause.

Example 5. Assume that we have derived the following clauses: $\langle a \vee b \vee \neg c\rangle, [\neg a \mid \{d\}], [c \mid \{\neg e\}]$. This means that there exists a closed tableau with root a in the branches containing d and a closed tableau with root $\neg c$ in the branches containing $\neg e$. Then it is clear that from these two tableaux and the clause above one can construct a closed tableau of root $\neg b$, if the current branch contains d and $\neg e$. Thus we derive $[b \mid \{d, \neg e\}]$.

Clash

$$\frac{\langle l_1 \vee \ldots \vee l_n\rangle, [l'_1 \mid \emptyset], \ldots, [l'_n \mid \emptyset]}{\langle \Box \rangle}$$

where $(l_1^c, \ldots, l_n^c)$ and $(l'_1, \ldots, l'_n)$ are unifiable.

The Clash rule is similar to the Propagation rule, but it applies when the root is reached. It only needs to be applied once.

Deletion

$$\frac{[l \mid \Phi \cup \{l'\}]}{[l \mid \Phi]\sigma}$$

where σ is the mgu of l', l^c.

The rôle of the Deletion rule is to simplify the constraint part of the *BTC*-formula $[l \mid \Phi]$ when possible. Intuitively, if a closed tableau of root l can be constructed from a set of literals Φ, then it can also be constructed from $\Phi \setminus \{l\}$ (since by definition l already occurs in the branch at root node).

Let S be a set of *BTC*-formula. We write $S \vdash \pi$ if π is deducible from S by a finite number of applications of the above rules.

The Deletion rule corresponds to the Reduction rule in connection tableaux. As we shall see the Deletion rule in never applied when Horn-renamable clause sets are considered. Note that the tableaux that are constructed using this procedure are always connection tableaux (by definition of the Propagation rule).

4 Soundness and Completeness

In this section we prove the basic properties of our proof procedure, namely soundness and refutational completeness.

As usual, an inference rule is said to be *sound* if the deduced *BTC*-formula is a logical consequence of the premisses.

Lemma 1. *The rules Link, Transitivity, Leaf Detection, Propagation, Clash and Deletion are sound.*

Proof. The proof is immediate for the rules Link, Transitivity, Leaf Detection since the inferred *BTC*-formulae are either of the form $(l \to k)$ or $[l \mid l]$, hence are true in any interpretation. Now we consider the remaining rules.

- **Propagation.** Let π be a *BTC*-formula deduced by the Propagation rule from a set of *BTC*-formulae S. Then π is necessarily of the form $[l \mid \Phi_1 \cup \ldots \cup \Phi_n]\sigma$, where S contains *BTC*-formulae of the form $\langle l_1 \vee \ldots \vee l_n \vee l\rangle$ and $[l'_i \mid \Phi_i]$ for any $i \in [1..n]$, and where σ is a unifier of $(l_1^c, \ldots, l_n^c)$ and $(l'_1, \ldots, l'_n)$.

  Assume that $I \models S$. Let θ be a ground substitution. We prove that $I \models \pi\theta$. Since $I \models S$, we have $I \models \langle l_1 \vee \ldots \vee l_n \vee l\rangle$. Thus there exists a literal m in $l_1 \vee \ldots \vee l_n \vee l$ such that $I \models m\sigma\theta$.

  We distinguish two cases.
  - If $m = l_j$, for some $j \in [1..n]$. Since $I \models S$ we have $I \models [l'_j \mid \Phi_j]$. But $l'_j\sigma = l_j^c\sigma$, hence since $I \models l_j\sigma\theta$ we have $I \not\models l'_j\sigma\theta$. By definition of the semantics of $[l'_j \mid \Phi_j]$ this implies that there exists a literal $m' \in \Phi_j$ such that $I \not\models m'\sigma\theta$. But $m'\sigma \in (\Phi_1 \cup \ldots \cup \Phi_n)\sigma$, thus $I \models \pi\theta$.
  - If $m = l$ then $I \models l\sigma\theta$ thus $I \models \pi\theta$.
- **Clash.** Assume that π is deduced by the Clash rule from a set of *BTC*-formulae S. Then S must contain *BTC*-formulae of the form $\langle l_1 \vee \ldots \vee l_n\rangle$, $[l'_1 \mid \emptyset], \ldots, [l'_n \mid \emptyset]$ and there exists a unifier σ of $(l_1^c, \ldots, l_n^c)$ and $(l'_1, \ldots, l'_n)$. Let I be an interpretation satisfying S. Let θ be a ground substitution. We have $I \models \langle l_1 \vee \ldots \vee l_n\rangle$, thus there exists $j \in [1..n]$ such that $I \models l_i\sigma\theta$. By definition we have $I \models [l'_i \mid \emptyset]$ for any $i \in [1..n]$, thus $I \models l'_i\sigma\theta$. But since σ is a unifier of l'_i and l_i^c we have $l'_i\sigma\theta = l_i^c\sigma\theta$ hence we get a contradiction (S is unsatisfiable).

- **Deletion.** Assume that π is deduced by the Deletion rule from a set of BTC-formulae S. Then $\pi = [l \mid \Phi]\sigma$ where S contains a BTC-formula $[l \mid \Phi \cup \{l'\}]$, where $l^c\sigma = l'\sigma$.

  Let θ be a ground substitution. If $I \models l\sigma\theta$ then we have $I \models \pi\theta$, by definition. Thus we assume that $I \not\models l\sigma\theta$. Then since $I \models [l \mid \Phi \cup \{l'\}]$, there exists a literal $m \in \Phi \cup \{l'\}$ such that $I \not\models m\sigma\theta$. If $m = l'$ then we have $I \not\models l'\sigma\theta$, which is impossible since $l'\sigma = l^c\sigma$ and $I \not\models l\sigma\theta$. Thus $m \neq l'$ and m must occur in Φ. Therefore we have $I \models \pi\theta$.

If S is a set of clauses, we denote by $\langle S\rangle$ the set of BTC-formulae: $\langle S\rangle \overset{\text{def}}{=} \{\langle C\rangle \mid C \in S\}$.

Theorem 1. *(Soundness) Let S be a set of clauses. If $\langle S\rangle \vdash \langle\Box\rangle$ then S is unsatisfiable.*

Proof. Let I be an interpretation satisfying S. By definition we have $I \models \langle S\rangle$. By Lemma 1 this implies that $I \models \langle\Box\rangle$ which is impossible.

Now we prove refutational completeness. We first show that BTC is complete for sets of ground BTC-formulae, then we use a lifting lemma to extend this result to the non ground level.

Let S be a set of ground clauses. Let l be literal. We denote by S_l the set of clauses obtained by interpreting l to *true*. More formally, S_l is obtained from S by removing each clause containing l and by deleting each occurrence of l^c in the remaining clauses.

In a first step, we ignore Condition 4 on the rule Link (we shall show afterwards that all blocked clauses can be removed).

Lemma 2. *Let S be a set of ground clauses. Let m be a literal occurring in a clause in S. Let π be a BTC-formula such that $\langle S_m\rangle \vdash \pi$. Assume that $\langle S\rangle \models [m \mid \Psi]$. If $\langle S\rangle \not\vdash \pi$, then either $\pi = \langle\Box\rangle$ and $\Psi \neq \emptyset$ or π is of the form $[l \mid \Phi]$ and $\langle S\rangle \vdash [l \mid \Phi \cup \Psi]$.*

Proof. The proof is by induction on the length of the derivation leading to π. We distinguish several cases, according to the rule used to derive π.

- **Link.** If π is deduced by the rule Link, then π is of the form $(l \rightarrow k)$, where $\langle S_m\rangle \vdash \langle l \vee k^c \vee C \vee l_1^c \vee \ldots \vee l_n^c\rangle, [l_1 \mid \phi_1], \ldots [l_n \mid \phi_n]$.

  By the induction hypothesis, we have $\langle S\rangle \vdash [l_1^c \mid \phi_1'], \ldots, [l_n^c \mid \phi_n']$, where ϕ_i' is either ϕ_i or $\phi_i \cup \Psi$. By definition, $l \vee k^c \vee C \vee l_1^c \vee \ldots \vee l_n^c$ occurs in S_m (since it is non empty, and the rules cannot derive any BTC-formula of the form $\langle D\rangle$ where $D \neq \Box$), thus S contains either $l \vee k^c \vee C \vee l_1^c \vee \ldots \vee l_n^c$, or $l \vee k^c \vee C \vee l_1^c \vee \ldots \vee l_n^c \vee m^c$.

  In both cases, since $[m \mid \Psi]$ is derivable, it is clear that the **Link** rule is applicable on the above clause and from BTC-formulae deducible from S, yielding $(l \rightarrow k)$.
- **Transitivity.** π is of the form $(l_1 \rightarrow l_3)$, where $\langle S_l\rangle \vdash (l_1 \rightarrow l_2)$, $l_2 \not\succ l_3$ and $\langle S_l\rangle \vdash (l_2 \rightarrow l_3)$. By the induction hypothesis, $\langle S\rangle \vdash (l_1 \rightarrow l_2), (l_2 \rightarrow l_3)$, thus by Transitivity $\langle S\rangle \vdash (l_1 \rightarrow l_3)$.
- **Leaf Detection.** The proof is similar to the one of the previous case.

- **Propagation.** π is of the form $[l \mid \Phi_1 \cup \ldots \cup \Phi_n]$ where $\langle S_m \rangle \vdash \langle l_1 \vee \ldots \vee l_n \vee l \rangle, [l_1^c \mid \Phi_1], \ldots, [l_n^c \mid \Phi_n]$.

  By the induction hypothesis, we have $\langle S \rangle \vdash [l_1^c \mid \Phi'_1], \ldots, [l_n^c \mid \Phi'_n]$, where for any $i \in [1..n']$, Φ'_i is either Φ_i or $\Phi_i \cup \Psi$.
  Moreover, since $l_1 \vee \ldots \vee l_n \vee l$ is non empty, it must occur in S_m.
  Hence S contains either $l_1 \vee \ldots \vee l_n \vee l$, or $l_1 \vee \ldots \vee l_n \vee l \vee m^c$.
  - If S contains $l_1 \vee \ldots \vee l_n \vee l$, then we have $\langle S \rangle \models \langle l_1 \vee \ldots \vee l_n \vee l \rangle$, hence we can apply the Propagation rule, yielding: $[l \mid \Phi'_1 \cup \ldots \cup \Phi'_n]$. If we have $\Phi'_i = \Phi_i$ for any $i \in [1..n]$ then we have $\langle S \rangle \vdash \pi$. Otherwise, we have $\langle S \rangle \vdash [l \mid \Phi_1 \cup \ldots \cup \Phi_n \cup \Psi]$.
  - If S contains $l_1 \vee \ldots \vee l_n \vee l \vee m^c$. Since $\langle S \rangle \models [m \mid \Psi]$ we can apply the rule Propagate, yielding $[l \mid \Phi'_1 \cup \ldots \Phi'_n \cup \Psi] = [l \mid \Phi_1 \cup \ldots \cup \Phi_n \cup \Psi]$.
- **Clash.** In this case, we must have $\Psi = \emptyset$ (otherwise the proof is obvious). Then the proof is similar to the previous one.
- **Deletion.** π must be of the form $[l \mid \Phi]$, where $\langle S_m \rangle \vdash [l \mid \Phi \cup \{l^c\}]$.
  By the induction hypothesis, we have $\langle S \rangle \vdash [l \mid \Phi']$, where Φ' is either $\Phi \cup \{l^c\}$ or $\Phi \cup \{l^c\} \cup \Psi$. In both cases, the Deletion rule applies and yields either $[l \mid \Phi]$ or $[l \mid \Phi \cup \Psi]$ hence the proof is completed.

A *BTC*-formula π is said to be *l-dominated* if π is of the form $(l \rightarrow k)$ or $[l \mid \Phi]$.

Lemma 3. *Let S be a set of ground clauses, and let m be a maximal literal occurring in a clause in S. For any literal l such that $\langle S \rangle \not\models (l \rightarrow m)$ and for any l-dominated BTC-formula π, if $\langle S_m \rangle \vdash \pi$ then $\langle S \rangle \vdash \pi$.*

Proof. The proof is by induction on the length of the derivation leading to π.

- **Link.** If π is deduced by the rule Link, then π is of the form $(l \rightarrow k)$, where $\langle S_m \rangle \vdash \langle l \vee k^c \vee C \vee l_1^c \vee \ldots \vee l_n^c \rangle, [l_1 \mid \phi_1], \ldots [l_n \mid \phi_n]$.

  S contains either $l \vee k^c \vee C \vee l_1^c \vee \ldots \vee l_n^c$ or $l \vee k^c \vee C \vee l_1^c \vee \ldots \vee l_n^c \vee m^c$. If the second case, since m is maximal, the Link rule is applicable yielding $(l \rightarrow m)$, and the proof is completed.

  Otherwise, assume that there exists a literal l_i in $l_1, \ldots, l_n$ such that $\langle S \rangle \not\vdash [l_i \mid \phi_i]$. We assume, w.l.o.g. that l_i is the greatest literal (according to $\succ$) having this property. By the induction hypothesis, we have $\langle S \rangle \vdash (l_i \rightarrow m)$. If $k \succ l_i$ then the Link rule is applicable on the above clause, yielding $(l \rightarrow k)$. Otherwise, the Link rule is applicable on the literals l and l_i^c, yielding $(l \rightarrow l_i)$. By Transitivity, this entails that $\langle S \rangle \vdash (l \rightarrow m)$ (since $l_i \prec m$) hence the proof is completed.

  Now assume that there for any $i \in [1..n]$, $\langle S \rangle \vdash [l_i \mid \phi_i]$. Then the rule Link is applicable and $(l \rightarrow k)$ is deducible.
- **Transitivity.** π is of the form $(l_1 \rightarrow l_3)$, where $\langle S_l \rangle \vdash (l_1 \rightarrow l_2)$ and $\langle S_l \rangle \vdash (l_2 \rightarrow l_3)$. Assume that $\langle S \rangle \not\models (l_1 \rightarrow m)$. By the induction hypothesis, $\langle S \rangle \vdash (l_1 \rightarrow l_2)$. If $\langle S \rangle \vdash (l_2 \rightarrow l_3)$, then the proof follows by Transitivity. Otherwise, by the induction hypothesis, we must have $\langle S \rangle \vdash (l_2 \rightarrow m)$, hence by Transitivity $\langle S \rangle \vdash (l_1 \rightarrow m)$ which is impossible.
- **Leaf Detection.** The proof is similar to the previous one.

- **Propagation.** π is of the form $[l \mid \Phi_1 \cup \ldots \cup \Phi_n]$ where $\langle S_m \rangle \vdash \langle l_1 \vee \ldots \vee l_n \vee l \rangle, [l_1^c \mid \Phi_1], \ldots, [l_n^c \mid \Phi_n]$.
  Assume that $\langle S \rangle \not\vdash (l \rightarrow m)$.
  S contains either $l_1 \vee \ldots \vee l_n \vee l$ or $l_1 \vee \ldots \vee l_n \vee l \vee m^c$. In the second case, since m is maximal, $(l \rightarrow m)$ is deducible. Thus $S \vdash \langle l_1 \vee \ldots \vee l_n \vee l \rangle$. Assume that there exists a literal l_i in $l_1, \ldots, l_n$ such that $\langle S \rangle \not\vdash [l_i^c \mid \Phi_i]$. W.l.o.g. we assume that l_i is the greatest literal having this property. Then it is clear that the Link rule is applicable, yielding $(l \rightarrow l_i^c)$. Then we must have $\langle S \rangle \not\vdash (l_i^c \rightarrow m)$ and by the induction hypothesis, $\langle S \rangle \models [l_i^c \mid \Phi_i]$ which is impossible.
  Thus we have $\langle S \rangle \vdash [l_1^c \mid \Phi_1], \ldots, [l_n^c \mid \Phi_n]$ and by Propagation $\langle S \rangle \vdash \pi$.
- **Clash.** The proof is obvious.
- **Deletion.** The proof is immediate.

Now we show that blocked clauses can be removed:

Lemma 4. *Let S be a set of ground clauses. Let C be a clause that is blocked in S. Then $S \setminus \{C\}$ is satisfiable iff S is satisfiable.*

Proof. Let M be a model of $S \setminus \{C\}$. If $M \models S$ then the proof is completed. Otherwise, M falsifies all literals in C.

By definition there exists a literal m such that C contains two literals l, k with $l^c \vartriangle_S m$ and $k^c \vartriangle_S m^c$. We have $M \models l^c, k^c$. We have either $M \models m$ or $M \models m^c$. By symmetry, we assume, w.l.o.g., that $M \models m$. Let M' be an interpretation such that $M' \models k$ and for any literal $x \notin \{k, k^c\}$ $M' \models x$ iff $M \models x$. Since $k^c \vartriangle_S m^c$ we have $m \neq k^c$. Moreover, since $M \models m$ and $M \not\models k$, we have $m \neq k$. Thus, since $M \models m$, we deduce that $M' \models m$. Since $M' \models k$, we have $M' \models C$. Assume that $M' \not\models S$. Then there exists a clause $D \in S$ such that $M' \not\models D$. Since $M \models D$, k^c must occur in D. But $k^c \vartriangle_S m^c$, thus (since $m \neq k$) m occurs in D. But $M' \models m$ hence $M' \models D$, which is impossible.

Lemma 5. *Let S be a finite set of* ground *clauses. If S is unsatisfiable then $\langle S \rangle \vdash \langle \Box \rangle$.*

Proof. A set of clauses S is said to be *minimally unsatisfiable* iff S is unsatisfiable and if for any clause $C \in S$, $S \setminus \{C\}$ is satisfiable. We prove, by induction on the number of distinct literals occurring in S, that if S is minimally unsatisfiable, then $\langle S \rangle \vdash \langle \Box \rangle$ **and** for any clause $l_1 \vee \ldots \vee l_n$ in S, we have $\langle S \rangle \vdash [l_i^c \mid \emptyset]$, for $i \in [1..n]$. By Lemma 4, we may assume that all the clauses in S are non-blocked.

Let l be a maximal literal in S. S_{l^c} and S_l are unsatisfiable and contain strictly less literals than S. Let S' be the minimally unsatisfiable subset of S_{l^c} and let S'' be the minimally unsatisfiable subset of S_l. Obviously, if $S' \subseteq S$ or $S'' \subseteq S$ then we have $S' = S$ or $S'' = S$ thus l (or l^c) does not occur in S which is impossible. Thus we have $S' \not\subseteq S$ and $S'' \not\subseteq S$.

S' contains a clause $l_1 \vee \ldots \vee l_n$ such that $l \vee l_1 \vee \ldots \vee l_n$ occurs in S. By the induction hypothesis, for any $i \in [1..n]$, we have $\langle S_{l^c} \rangle \vdash [l_i^c \mid \emptyset]$. Similarly S'' contains at least one clause $m_1 \vee \ldots \vee m_k$ such that $l^c \vee m_1 \vee \ldots \vee m_k$ occurs in S. By the induction hypothesis $\langle S_l \rangle \vdash [m_i^c \mid \emptyset]$ for any $j \in [1..k]$.

If $[l \mid \emptyset]$ or $[l^c \mid \emptyset]$ is deducible, then since $S_l \vdash \langle \Box \rangle$ and $S_{l^c} \vdash \langle \Box \rangle$, we must have $\langle S \rangle \vdash \langle \Box \rangle$ by Lemma 2. Otherwise, we distinguish two cases.

- Assume that $\langle S \rangle \not\vdash [l \mid l]$. Then we have either $\langle S \rangle \not\vdash (l \rightarrow l^c)$, or $\langle S \rangle \not\vdash (l^c \rightarrow l)$ or $l \not\in \Gamma$ and $\langle S \rangle \vdash [l^c \mid l^c]$ (by the rule Leaf Detection). The case $\langle S \rangle \vdash [l^c \mid l^c]$ is handled below (replacing l^c by l). Thus we assume, w.l.o.g. that $\langle S \rangle \not\vdash (l \rightarrow l^c)$.

  Assume that there exists a literal l_i in $l_1, \ldots, l_n$ such that $\langle S \rangle \vdash (l_i^c \rightarrow l^c)$. Let l_i be the greatest literal having this property. For any j such that $l_j \succ l_i$ we have $\langle S \rangle \not\vdash (l_j \rightarrow l^c)$ thus by Lemma 3 $\langle S \rangle \vdash [l_j^c \mid \emptyset]$. Then the Link rule applies yielding $(l \rightarrow l_i^c)$, and by Transitivity (since $l \succ l_i$) we have $\langle S \rangle \vdash (l \rightarrow l^c)$ which is impossible.

  Thus there is no literal l_i in $l_1, \ldots, l_n$ such that $\langle S \rangle \vdash (l_i^c \rightarrow l^c)$, hence by Lemma 3, this implies that for any $i \in [1..n]$, $\langle S \rangle \vdash [l_i^c \mid \emptyset]$. Thus by Propagation we get $[l \mid \emptyset]$.
- Assume that $\langle S \rangle \vdash [l \mid l]$. We have $\langle S_{l^c} \rangle \vdash [l_i^c \mid \emptyset]$, for any $i \in [1..n]$. By Lemma 2 since $\langle S \rangle \vdash [l \mid l]$, this implies that $\langle S \rangle \vdash [l_i^c \mid \Psi_i]$ $(i = 1, \ldots, n)$ where Ψ_i is either $\emptyset$ or $\{l\}$. By propagation, we get $\langle S \rangle \vdash [l \mid \Psi]$, where $\Psi = \bigcup_{i=1}^n \Psi_i$. By deleting, this gives $\langle S \rangle \vdash [l \mid \emptyset]$.

Now consider an arbitrary clause D in S. It is clear that either D occurs in $S' \cap S''$ or D is of the form $l \vee D'$ where $D' \in S'$ or D is of the form $l^c \vee D''$ where $D'' \in S''$. If D occurs in $S' \cap S''$ then for any $m \in D$, we have $\langle S' \rangle \vdash [m^c \mid \emptyset]$. Since $S \vdash [l \mid \emptyset]$ this implies that $\langle S \rangle \vdash [m^c \mid \emptyset]$. Then same holds if D is of the form $l \vee D'$ or $l^c \vee D''$ where $D' \in S'$ (resp. $D'' \in S''$), since we have shown that $\langle S \rangle \vdash [l^c \mid \emptyset], [l \mid \emptyset]$.

The non ground case is obtained by using a lifting lemma very close to the ones used for resolution calculi.

Lemma 6 *(Lifting Lemma)***.** *Let S be a set of BTC-formulae. Let S' a set of ground instances of S. If $S' \vdash \pi$, then there exist a BTC-formula π' and a substitution σ such that $S \vdash \pi'$ and $\pi = \pi'\sigma$.*

Corollary 1 *(Refutational Completeness)***.** *Let S be a set of clauses. If S is unsatisfiable then $\langle S \rangle \vdash \langle \Box \rangle$.*

We can also use the information deduced during the search in order to simplify the considered clause set. If a *BTC*-formulae of the form $[l \mid \emptyset]$ is generated, where l subsumes a clause C in S, then C is a logical consequence of $[l \mid \emptyset]$ thus $\langle C \rangle$ and all the *BTC*-formulae deduced from $\langle C \rangle$ (and their descendants) can be deleted. It is obvious that this strategy preserves refutational completeness (since the number of clauses strictly decreases at each application of the simplification rule). Of course, this strategy may introduce some redundancy, because the same *BTC*-formula can be deduced $|S|$ times, where $|S|$ is the size of the initial clause set (still if the number of deducible *BTC*-formulae is polynomial w.r.t. to $|S|$ then it is still polynomial when the deletion strategy is used).

5 Complexity

A good way of assessing the efficiency of a proof procedure (beside practical experimentations) is to investigate its behavior on some particular propositional classes.

In this section we prove that BTC (more precisely, the algorithm obtained by the non deterministic application of the above inference rules on the set of formula $\langle S \rangle$, where S denotes the set of clauses at hand) runs in polynomial time for three classes of propositional clause sets: the class of **q-Horn renamable clauses**, the class of **ordered renamable Horn clauses** and the **class $\mathbb{S}_0$** (with renaming). For the last class, an appropriate strategy is needed, defined by the pair $\succ, \Gamma$. All these classes are distinct strict extensions of the Horn class. These results are of course interesting by themselves, because it is useful to have a uniform procedure for all these classes, but also because they provide some hints of the power of our method in the general case (by showing how our approach can keep the search space compact). To the best of our knowledge, there is no other proof procedure sharing these properties.

A *renaming* r is a function mapping each literal l to either l or l^c, such that $r(l^c) = r(l)^c$. A renaming can be extended to clauses and clause sets as follows: $r(l_1 \vee \ldots \vee l_n) \stackrel{\text{def}}{=} r(l_1) \vee \ldots \vee r(l_n)$, and $r(S) \stackrel{\text{def}}{=} \{r(C) \mid C \in S\}$.

A clause is *Horn* if it contains at most one positive literal and *Krom* if it contains at most most two literals.

5.1 q-Horn Class

The class of q-Horn clauses has been introduced in [7]. Let X be a set of atoms. A literal is said to be *on X*, if either $l \in X$ or $l^c \in X$.

Definition 1. *Let S be a set of ground clauses. S is* q-Horn *if there exists two sets of atoms X and Y such that all the clauses C in S satisfy one of the following conditions:*

- *C is a Horn clause and all literals in C are on X.*
- *All literals in C are on $X \cup Y$, C contains no positive literal on X and at most two literals on Y.*

In particular, any set of Horn clauses and any set of Krom clauses is q-Horn. A clause set S is said to be *q-Horn renamable* iff there exists a renaming r such that $r(S)$ is q-Horn.

Theorem 2. *Let S be a q-Horn renamable set of ground clauses. The number of BTC-formulae π such that $\langle S \rangle \vdash \pi$ is at most $o(a^3)$, where a denotes the number of distinct atoms occurring in S.*

Proof. Let r be a renaming r such that $r(S)$ is q-Horn. Let X, Y be the corresponding sets of atoms. We first show that we can infer no BTC-formula of the form $(p \rightarrow l)$ where $r(p) \in X$ and $r(l) \notin X$.

Let $\pi = (p \rightarrow l)$ be such a BTC-formula. We assume, w.l.o.g., that π is chosen in such a way that the derivation leading to π is minimal.

- Assume that π is generated by Transitivity. Then there exists two *BTC*-formulae of the form $(p \to k)$ and $(k \to l)$ such that $\langle S \rangle \vdash (p \to k)$ and $\langle S \rangle \vdash (k \to l)$. By definition of π, $r(k) \in X$ (otherwise, $(p \to k)$ would satisfy the above property hence the derivation leading to π would not be minimal). Thus $(k \to l)$ satisfies the above property which is impossible.
- Otherwise, π is generated by Link. Then there exists in S a clause of the form $p \vee l^c \vee C$. Since $r(p) \in X$, $r(p \vee l^c \vee C)$ is Horn and contains no literals on Y. Thus $r(l) \in X$.

We now show (by induction on the length of the derivation) that if $\langle S \rangle \vdash (l \to k)$ and l is a literal on Y and k a literal on X, then $r(k)$ is positive.

- Assume that $(l \to k)$ is generated by Transitivity. Then $\langle S \rangle \vdash (l \to l'), (l' \to k)$. If l' is a literal on Y, then by the induction hypothesis, $r(k)$ is positive. Otherwise, l' must be positive. Then by the above property, $r(k)$ is an atom on X.
- Otherwise, $(l \to k)$ is generated by Link. Then S contains a clause $l \vee k^c \vee C$. Since l is on Y and k is on X, $r(k)$ must be positive.

Then, we show, by induction on the derivation, that if a *BTC*-formula $[l \mid \Phi]$ is generated, then either $r(l)^c \in X$ and Φ contains at most two literals (from Y) or $r(l) \in Y$ and $\Phi = \emptyset$ or l is on Y and Φ is of the form $\{k\}$ where k is a literal on Y.

- If $[l \mid \Phi]$ is generated by the rule Leaf Detection, then by definition we have $\Phi = \{l\}$ where $\langle S \rangle \vdash (l \to l^c), (l^c \to l)$. If l is on Y then the proof is completed. Thus we assume that l is on X. If $r(l)$ is positive, then we cannot have $\langle S \rangle \vdash (l \to l^c)$ since $r(l^c) \notin X$. Similarly, if $r(l)$ is negative, then we have $\langle S \rangle \not\vdash (l^c \to l)$.
- If $[l \mid \Phi]$ is obtained by the rule Propagate, then there exists a clause of the form $l \vee l_1 \vee \ldots \vee l_n$ such that $\langle S \rangle \vdash [l_1^c \mid \Phi_1], \ldots, [l_n^c \mid \Phi_n]$ where $\Phi = \bigcup_{i=1}^n \Phi_i$. We distinguish two cases.
  - Either $l \vee l_1 \vee \ldots \vee l_n$ contains no literal on Y. Then $r(l \vee l_1 \vee \ldots \vee l_n)$ is Horn. If $r(l)$ is positive then all the $r(l_i)$'s are negative. By the induction hypothesis, Φ_i must be empty, thus Φ is empty.

    If $r(l)$ is negative, then there exists at most one $i \in [1..n]$ such that $r(l_i)$ is positive. By the induction hypothesis, there exists at most one $i \in [1..n]$ such that $\Phi_1, \ldots, \Phi_n$ is non-empty and Φ_i contains at most two literals in Y. Thus Φ contains two literals in Y.
  - Or $r(l \vee l_1 \vee \ldots \vee l_n)$ contains at most two literals on Y and only negative literals on X. If l is on X, then $r(l)$ is negative. $l_1, \ldots, l_n$ contains (at most) two literals on Y, say l_1, l_2. By the induction hypothesis, all the Φ_i for $i = 3, \ldots, n$ are empty. Moreover, we have $\Phi_1 = \{k_1\}$, $\Phi_2 = \{k_2\}$ where k_1, k_2 are on Y. Thus Φ contains at most two literals in Y.

    Otherwise, l is on Y. Moreover, all the literals on X occurring in $r(l_1)$, $\ldots, r(l_n)$ are negative and there is at most one literal on Y occurring in $l_1, \ldots, l_n$. Thus there is at most one $i \in [1..n]$ such that Φ_i is non empty and Φ_i must be a singleton. Thus Φ is either empty or a singleton.

Therefore, all the *BTC*-formulae that can be generated (beside $\langle \Box \rangle$) are either of the form $(l \to k)$ where l, k are literals or of the form $[l \mid \Phi]$ where l is a literal and Φ is a set of literals of cardinality 2 or less. Obviously, there exist at most $o(a^3)$ such formulae, where a denotes the number of ground atoms.

5.2 Ordered Clauses

We now consider another extension of the Horn class, the class of *ordered clauses* [4]. It strictly contains some other polynomial classes such as the Simple Enlarged Horn clauses [16] or the Simple Extended Horn clauses [9] (see also [15]).

We denote by $occ_S(l)$ the set of clauses $C \in S$ such that l occurs in S. We write $l \sqsubset_S k$ if $occ_S(l^c) \subseteq occ_S(k^c)$. Note that if $l \sqsubset_S k$ and $l \neq k$, then $l^c \vartriangle_S k$, by definition of $\vartriangle_S$.

A literal is called *bounded* in C (w.r.t. S) if either $occ_S(l^c) = \emptyset$ or if there exists a literal $k \in C$ such that $l \neq k$ and $l \sqsubset_S k$. A non bounded literal is said to be *free.* A set of clauses S is said to be *ordered* if each clause in S contains at most one free positive literal. A clause set S is said to be *ordered renamable* iff there exists a renaming r such that $r(S)$ is ordered.

Theorem 3. *Let S be a ordered renamable set of ground clauses. Then the number of BTC-formulae π such that $\langle S \rangle \vdash \pi$ is at most $o(a^2)$ where a denotes the number of ground atoms in S.*

Proof. Let r be a renaming such that $r(S)$ is ordered. We show that there is no pair of literals p, q such that $\langle S \rangle \vdash (p \to \neg q)$ and $r(p), r(q)$ are positive.

It is easy to see that the only rule that can generate the *first* literal of this form is Link. Assume that there exists a clause $p \vee q \vee C$ from which $(p \to \neg q)$ can be deduced. Then since C is ordered, either $r(p)$ or $r(q)$ must be bounded. By the application conditions on the rule Link, $p \vee q \vee C$ is non blocked. We distinguish several cases.

- $occ_{r(S)}(r(q)^c) = \emptyset$. Then $occ_S(q^c) = \emptyset$, thus $q^c \vartriangle_S p$ holds by definition of $\vartriangle_S$. Since $p \vartriangle_S p$, this entails that C is blocked and thus contradicts the application condition of Link.
- $occ_{r(S)}(r(p)^c) = \emptyset$. Then $p^c \vartriangle_S q$ holds. Since $q \vartriangle_S q$, C is blocked, which is impossible.
- There exists a literal s in C, such that $r(s)$ is positive, s is distinct from p, q and $r(q) \sqsubset_{r(S)} r(s)$. Then it is clear that this implies $(\neg q) \vartriangle_S s$. Since $s \vartriangle_S s$, this entails that C is blocked.
- There exists a literal s in C such that $r(s)$ is positive, distinct from p, q and $r(p) \sqsubset_{r(S)} r(s)$. Then we have $(\neg p) \vartriangle_S s$, and C is blocked.
- $r(p) \sqsubset_S r(q)$. Then we have $(\neg r(p)) \vartriangle_{r(S)} r(q)$, hence $(\neg p) \vartriangle_S q$, thus C is blocked.
- $r(p) \sqsubset_{r(S)} r(p)$. Then $(\neg q) \vartriangle_S p$, hence C is blocked.
- Otherwise, $r(p)$ and $r(q)$ are both free in $r(S)$, which is impossible since $r(C)$ is ordered and $r(p), r(q)$ are positive.

Therefore, no BTC-formula of the form $(p \to \neg p)$, where $r(p)$ is positive, can be deduced and the Leaf Detection rule is never applied. This entails that for any deducible BTC-formula of the form $[l \mid \Phi]$, we have $\Phi = \emptyset$. Thus there is only $o(a^2)$ possible BTC-formulae, where a denotes the number of distinct ground atoms in S.

5.3 The Class $\mathbb{S}_0$

The class $\mathbb{S}_0$ is introduced in [17]. We slightly extend the definition to handle renaming. A set of clauses $\{C_1, \dots, C_n\}$ belongs to the class $\mathbb{S}_0$ if there exists a renaming r such that the following holds:

1. For any $i = 1, \dots, n$, $C_i = P_i \cup H_i$, where $r(H_i)$ is a Horn clause and $r(P_i)$ is a set of positive literals.
2. For any $i = 1, \dots, n-1$, $P_{i+1} \subseteq P_i$.

In contrast to the previous cases, in order to handle this class properly we need to impose additional constraints on the strategy. We assume that $\succ$ and $\varGamma$ satisfy the following properties: For any $i \in [1..n]$, if $p \in P_i$ and $q \notin P_i$, then $p \succ q$; and $p \in \varGamma$ iff $r(p)$ is positive.

Theorem 4. *Let S be a set of ground clauses in $\mathbb{S}_0$. Then the number of BTC-formulae π such that $\langle S \rangle \vdash \pi$ is at most $o(a^2)$ where a denotes the number of ground atoms in S.*

Proof. (Sketch) We show that there is no pair of literals p, q such that $[p \mid \{p\}]$ and $[q \mid \{q\}]$ are both derivable, and either $p \neq q$ or $p \notin P_1$. The proof is by induction on the length of the derivation. Assume that such a pair of literals exists. By symmetry, we assume $p \preceq q$.

By the above strategy, $r(p)$ and $r(q)$ are both positive. By definition $[p \mid \{p\}]$ must have been deduced by the rule Link. Thus $(p \rightarrow \neg p)$ must be derivable. By definition, this implies that there exists a sequence of literals $l_1, \dots, l_n$ such that $l_1 = p$, $l_n = \neg p$ and for any $i = 1, \dots, n-1$, $(l_i \rightarrow l_{i+1})$ is deducible from $\varGamma$ by applying the rule Link. Moreover, by the application condition on the rule Transitivity we have $l_i \prec p$, for any $i \in [2..n-1]$. Let i be the first index in $[1..n]$ such that $r(l_i)$ is negative. Let $a = l_{i-1}$, $b = l_i^c$. By definition, S contains a clause $C_j = (a \vee b \vee D)$ such that $(a \rightarrow \neg b)$ is derivable from $\langle C_j \rangle$ using Link. By definition, a or b must be in P_j. Moreover, $a, b \preceq p$. Thus p is in $P_j \subseteq P_1$. Then this implies that $q \succ p$, hence $q \in P_j$. Therefore, q must occur in C_j. By definition of the rule Link, $\varGamma$ contains a *BTC*-formula of the form $[q^c \mid \varPhi]$. Since $[q \mid q]$ is deducible, $\varPhi$ cannot be empty (otherwise $[q^c \mid \emptyset]$ would have been deduced *before* $[q \mid q]$ which is impossible because due to deletion strategy introduced at the end of Section 4 any literal of the form $(l^c \rightarrow \dots)$ would be deleted). By the induction hypothesis, we have $\varPhi \subseteq \{q\}$. But q occurs in C_j which is impossible (by definition of the Link rule).

This implies that, at a given point, only one *BTC*-formula of the form $[l \mid \{l\}]$ exists. Thus the constraint parts of the *BTC*-formulae contain at most one literal and at most $o(a^2)$ distinct *BTC*-formulae can be generated.

6 Conclusion

A new proof procedure for first-order logic, called BTC, has been presented. Its soundness and refutational completeness has been proven. We have also investigated the complexity of BTC on some propositional classes and showed that it is polynomial for all these classes.

A natural continuation of this work is to implement this procedure in order to estimate its practical performances. The extension of this approach to first-order logic with equality should also be considered, and refinements can be defined in order to prune the search space (for instance it is obvious that a form of subsumption can be used to delete redundant *BTC*-formulae).

References

1. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper-tableaux. In: Orłowska, E., Alferes, J.J., Moniz Pereira, L. (eds.) JELIA 1996. LNCS, vol. 1126, Springer, Heidelberg (1996)
2. Beckert, B.: Depth-first proof search without backtracking for free-variable clausal tableaux. Journal of Symbolic Computation 36, 117–138 (2003)
3. Beckert, B., Posegga, J.: Lean-TAP: Lean tableau-based deduction. Journal of Automated Reasoning 15(3), 339–358 (1995)
4. Benoist, E., Hébrard, J.J.: Ordered formulas. Technical Report 14, Université de Caen, Les cahiers du CGREYC (1999)
5. Bibel, W.: On matrices with connections. Journal of the Association of Computing Machinery 28, 633–645 (1981)
6. Billon, J.-P.: The disconnection method: a confluent integration of unification in the analytic framework. In: Miglioli, P., Moscato, U., Ornaghi, M., Mundici, D. (eds.) TABLEAUX 1996. LNCS, vol. 1071, Springer, Heidelberg (1996)
7. Boros, E., Crama, Y., Hammer, P.L.: Polynomial-time inference of all valid implications for Horn and related formulae. Ann. Mathematics and Artificial Intelligence 1, 21–32 (1990)
8. Bry, F., Yahya, A.: Minimal model generation with positive unit hyper-resolution tableaux. In: Miglioli, P., Moscato, U., Ornaghi, M., Mundici, D. (eds.) TABLEAUX 1996. LNCS, vol. 1071, Springer, Heidelberg (1996)
9. Chandru, V., Hooker, J.N.: Extended horn sets in propositional logic. J. ACM 38(1), 205–221 (1991)
10. Giese, M.: Incremental Closure of Free Variable Tableaux. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, Springer, Heidelberg (2001)
11. Hähnle, R.: Tableaux and related methods. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, Elsevier Science vol. I, ch. 3, pp. 100–178 (2001)
12. Letz, R., Stenz, G.: Model elimination and connection tableau procedures. pp. 2015–2112 (2001)
13. Letz, R., Stenz, G.: Proof and model generation with disconnection tableaux. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, Springer, Heidelberg (2001)
14. Peltier, N.: Some techniques for branch-saturation in free-variable tableaux. In: Alferes, J.J., Leite, J.A. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, Springer, Heidelberg (2004)
15. Schlipf, J.S., Annexstein, F.S., Franco, J.V., Swaminathan, R.P.: On Finding Solutions for Extended Horn Formulas. Information Processing Letters 54(3), 133–137 (1995)
16. Swaminathan, R.P., Wagner, D.K.: The arborescence-realization problem. Discrete Appl. Math 59(3), 267–283 (1995)
17. Yamasaki, S., Doshita, S.: The satisfiability problem for a class consisting of Horn sentences and some non-Horn sentences in propositional logic. Information and Control 59, 1–12 (1983)

Differential Dynamic Logic for Verifying Parametric Hybrid Systems[⋆]

André Platzer

University of Oldenburg, Department of Computing Science, Germany
Carnegie Mellon University, Computer Science Department, Pittsburgh, PA
platzer@informatik.uni-oldenburg.de

Abstract. We introduce a first-order dynamic logic for reasoning about systems with discrete and continuous state transitions, and we present a sequent calculus for this logic. As a uniform model, our logic supports hybrid programs with discrete and differential actions. For handling real arithmetic during proofs, we lift quantifier elimination to dynamic logic. To obtain a modular combination, we use side deductions for verifying interacting dynamics. With this, our logic supports deductive verification of hybrid systems with symbolic parameters and first-order definable flows. Using our calculus, we prove a parametric inductive safety constraint for speed supervision in a train control system.

Keywords: dynamic logic, sequent calculus, verification of parametric hybrid systems, quantifier elimination.

1 Introduction

Frequently, correctness of a real-time or hybrid system [16] depends on the choice of parameters [9, 12, 24]. Such parameters naturally arise from the degrees of freedom of how a part of the system can be instantiated or how a controller can respond to input. They include both external system parameters like the braking force of a train, and control parameters of internal choice like when to start braking before approaching an open gate or a preceding train [9].

Symbolic parameters occurring in system dynamics raise a couple of challenges. Even simple parametric flows and guards are *non-linear*: With parameter b, the flow constraint $2x + by \geq 0$ is an algebraic inequality but not linear. For this reason, we cannot use approaches with linear arithmetic like the model checkers HyTech [1], or PHAVer [14]. More generally, Davoren and Nerode [11] argue that, unlike deductive methods, model checking [1, 7, 14, 16] does not support free parameters. Furthermore, correctness of parametric systems typically depends on a constraint on the free parameters, which is not always known a

[⋆] This research was supported by a fellowship of the German Academic Exchange Service (DAAD) and by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, see www.avacs.org).

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 216–232, 2007.

priori but needs to be identified during the analysis. It is, however, quite complicated to synthesise such general symbolic constraints from the concrete values of a counterexample trace produced by a model checker. Even without parameters, there are limitations of non-symbolic techniques for handling continuous flows [24].

To overcome these issues, we propose a fully symbolic technique following a deductive approach. We introduce a logic for verifying hybrid systems with parameters. As a basis we use first-order logic, which has widely proven its power and flexibility in handling symbolic parameters as logical variables. For reasoning about state transitions, our logic further extends dynamic logic.

First-order dynamic logic (DL) [15] is a successful approach for reasoning about (discrete) state changes [3,4,15,18]. Like model checking, first-order DL can analyse the behaviour of operational system models [21,22]. Yet, DL calculi accept parameters: they verify systems by deductive proof rather than a more enumerative and graph-theoretic analysis of the (abstract) state space as in model checking [7]. In addition, operational models are internalised, and DL is closed under logical operators. Thus, DL can analyse the relationship between multiple systems [21,23], which is useful for compositional verification.

Since hybrid systems are subject to both continuous evolution and discrete state change, we add continuous state changes to discrete DL. As a uniform model for hybrid systems, our logic introduces hybrid programs with discrete assignments and differential actions. The resulting first-order dynamic logic is called differential dynamic logic ($\mathsf{d}\mathcal{L}$). It has been motivated and proposed (without a formal introduction) in our preliminary work [21].

In [22,23], we have introduced logics that extend the basic ideas of [21] into different directions. In [23], we have presented a logic with nominals to investigate compositionality. In [22], we have introduced a temporal logic and a calculus that reduces temporal statements to non-temporal formulas. The calculi of [22,23] reduce their respective extensions to $\mathsf{d}\mathcal{L}$. In this paper, we give a proof system for $\mathsf{d}\mathcal{L}$ itself, which can be combined with the extensions in [22,23].

We show how interacting discrete and continuous dynamics can be verified *constructively* by integrating quantifier elimination into our calculus to handle the resulting arithmetic. Moreover, this combination is modular in the sense that we directly combine quantifier elimination and dynamic logic side by side. Using side deductions, we achieve such a modular combination even though both quantifiers and dynamic modalities interact by affecting the values of variables.

As an important modelling characteristic of hybrid systems, we generalise differential actions to differential equations that can be restricted to first-order invariant regions. Moreover, we show how induction can be integrated.

The first contribution of this paper is a formal introduction of our logic $\mathsf{d}\mathcal{L}$. The main contribution is a *full* calculus for verifying *interacting* discrete and continuous dynamics including the resulting arithmetic. For this, we present a modular combination of quantifier elimination with a sequent calculus. Using our calculus, we prove safety of speed supervision in train control [9,12] and synthesise the required parametric induction invariant.

Hybrid Systems. The behaviour of safety-critical systems typically depends on both the state of a discrete controller and continuous physical quantities. Hybrid systems are mathematical models for dynamic systems with interacting discrete and continuous behaviour [11,16]. Their behaviour combines continuous evolution (called *flow*) characterised by differential equations and discrete jumps.

Dynamic Logic. The principle of dynamic logic is to combine system operations and correctness statements about system states within a single specification language (see [15] for a general introduction for discrete systems). By permitting system operations α as actions of a labelled multi-modal logic, dynamic logic provides formulas of the form $[\alpha]\phi$ and $\langle\alpha\rangle\phi$, where $[\alpha]\phi$ expresses that all (terminating) runs of system α lead to states in which condition ϕ holds. Likewise, $\langle\alpha\rangle\phi$ expresses that there is at least one (terminating) run of α after which ϕ holds. In $\mathsf{d}\mathcal{L}$, hybrid programs play the role of α. In particular, $\mathsf{d}\mathcal{L}$ extends discrete dynamic logic [15] such that α can display continuous evolution.

Related Work. Several approaches [2,13,19,20] use quantifier elimination [8] in first-order real arithmetic for model checking hybrid automata. Thus, we use the same decision procedure as a basis for handling arithmetic of non-linear flows. We generalise these results to a deductive calculus for improved handling of free parameters. As a more uniform model that is amenable to compositional symbolic processing by calculi, we use hybrid programs rather than automata.

Zhou et al. [27] extended duration calculus with mathematical expressions in derivatives of state variables. They use a multitude of rules and an oracle that requires external mathematical reasoning about derivatives and continuity.

Rönkkö et al. [25] presented a guarded command language with differential relations and gave a semantics in higher-order logic with built-in derivatives. Without providing a means for verification of this higher-order logic, the approach is still limited to providing a notational variant of classical mathematics.

Rounds [26] defined a semantics in set theory of a "spatial" logic for a hybrid π-calculus. Without giving a calculus for that logic, this approach is not suitable for verification yet. Further, the automatic proving potential is limited by the large number of very expressive operators in this formalism.

Boulton et al. [6] introduced a Hoare calculus for gain and phase shift properties of a special case of block diagrams. It requires manual reasoning about complicated expressions with square roots, rational and trigonometric functions. The authors do not give a soundness result or formal semantics.

Davoren and Nerode [10,11] extended the propositional modal μ-calculus with a semantics in hybrid systems and examine topological aspects. They provided Hilbert-style calculi to prove formulas that are valid for all hybrid systems simultaneously. Thus, only limited information can be obtained about a particular system: In propositional modal logics, system behaviour needs to be axiomatised in terms of abstract actions a, b, c of *unknown effect*, see, e.g. [21].

The strength of our logic primarily is that it is a *first-order* dynamic logic: It handles actual operational models of hybrid systems like $x := x + y; \dot{x} = 2y$ rather than abstract propositional actions of unknown effect. Further, we provide a calculus for actually verifying hybrid programs with free parameters and first-order definable flows, i.e., flows that are definable in first-order arithmetic. For verifying the coordination level of train dynamics, which we use as an example, first-order definable flows are sufficient [9]. First-order approximations of more general flows can be used according to [2, 24].

Structure of this Paper. After introducing syntax and semantics of the differential logic $\mathsf{d}\mathcal{L}$ in Section 2, we introduce a sequent calculus in Section 3, which can be used for verifying parametric hybrid systems in $\mathsf{d}\mathcal{L}$, and prove soundness. In Section 4, we prove an inductive safety property in train control using the $\mathsf{d}\mathcal{L}$ calculus. Finally, we draw conclusions and discuss future work in Section 5.

2 Syntax and Semantics of Differential Logic

As a uniform model for hybrid systems, our logic introduces hybrid programs, which generalise real-time programs [17] to hybrid change. They are more amenable to step-wise symbolic processing by calculus rules than graph structures of automata. Since hybrid automata [1] can be embedded, there is no loss of expressivity. Hybrid programs have a simple compositional semantics. With this basis, we can construct a compositional calculus that reduces verification of system properties to proving properties of its parts.

The differential logic $\mathsf{d}\mathcal{L}$ is a dynamic logic for reasoning about programs with three basic characteristics to meet the requirements of hybrid systems:

Discrete jumps. Projections in state space are represented as instantaneous assignments of values to state variables. With this, mode switches like $\mathsf{mode} := 4$ or $\mathsf{signal} := 1$ can be expressed with discrete jumps, as well as resets $z := 0$ or adjustments of control variables like $z := z - 2$.

Continuous evolution. Continuous variation in system dynamics is represented with differential equations as evolution constraints. For example, the evolution of a train with constant braking can be expressed with a system action for the differential equation $\ddot{z} = -b$ with second time-derivative $\ddot{z}$ of z. Similarly, the effect of $\ddot{z} = -b \,\&\, z \geq 5$ is an evolution that is restricted to always remain within the region $z \geq 5$, i.e., to stop braking at the latest when $z < 5$. Such an evolution can stop at any time within $z \geq 5$, it can even continue with transient grazing along the border $z = 5$, but it is not allowed to proceed when it enters $z < 5$.

Regular combinations. Discrete and continuous transitions can be combined to form *hybrid programs* using regular expression operators $(\cup, {}^*, ;)$ as structured behaviour of hybrid systems. For example, $\mathsf{mode} := 4 \cup \ddot{z} = -b$ describes a train controller that can either choose to switch its state to an alert mode (4) or initiate braking by the differential equation $\ddot{z} = -b$, by a nondeterministic choice $(\cup)$. In conjunction with other regular combinations, control constraints can be expressed using conditions like $?z \geq 9$ as guards for the system state.

2.1 Syntax of d$\mathcal{L}$

Terms and Formulas. The formulas of d$\mathcal{L}$ are built over a finite set V of real-valued variables and a fixed signature Σ of function and predicate symbols. For simplicity, the signature Σ is assumed to contain exclusively the usual function and predicate symbols for real arithmetic, such as $0, 1, +, \cdot, =, \leq, <, \geq, >$.

The set $\mathrm{Trm}(V)$ of *terms* is defined as in classical first-order logic yielding polynomial expressions. The set $\mathrm{Fml}(V)$ of *formulas* of d$\mathcal{L}$ is defined as common in first-order dynamic logic [15]. That is, they are built using propositional connectives $\wedge, \vee, \rightarrow, \leftrightarrow, \neg$ and quantifiers $\forall, \exists$ (first-order part). In addition, if ϕ is a d$\mathcal{L}$ formula and α a hybrid program, then $[\alpha]\phi$, $\langle\alpha\rangle\phi$ are formulas (dynamic part).

Hybrid Programs. In d$\mathcal{L}$, elementary discrete jumps and continuous evolutions interact using regular control structure to form hybrid programs.

Definition 1 (Hybrid programs). *The set* $\mathrm{HP}(V)$ *of* hybrid programs *is inductively defined as the smallest set such that:*

- *If* $x \in V$ *and* $\theta \in \mathrm{Trm}(V)$*, then* $(x := \theta) \in \mathrm{HP}(V)$*.*
- *If* $x \in V$*,* $\theta \in \mathrm{Trm}(V)$*, and* $\chi \in \mathrm{Fml}(V)$ *further is a quantifier-free first-order formula, then* $(\dot{x} = \theta \,\&\, \chi) \in \mathrm{HP}(V)$*.*
- *If* $\chi \in \mathrm{Fml}(V)$ *is a quantifier-free first-order formula, then* $(?\chi) \in \mathrm{HP}(V)$*.*
- *If* $\alpha, \gamma \in \mathrm{HP}(V)$ *then* $(\alpha; \gamma) \in \mathrm{HP}(V)$*.*
- *If* $\alpha, \gamma \in \mathrm{HP}(V)$ *then* $(\alpha \cup \gamma) \in \mathrm{HP}(V)$*.*
- *If* $\alpha \in \mathrm{HP}(V)$ *then* $(\alpha^*) \in \mathrm{HP}(V)$*.*

The effect of $x := \theta$ is an instantaneous discrete jump in state space. That of $\dot{x} = \theta \,\&\, \chi$ is an ongoing continuous evolution controlled by the differential equation $\dot{x} = \theta$ while remaining within the region described by χ. The evolution is allowed to stop at any point in χ. It is, however, obliged to stop before it leaves χ. For unrestricted evolution, we abbreviate $\dot{x} = \theta \,\&\, \mathit{true}$ by $\dot{x} = \theta$. The d$\mathcal{L}$ semantics allows differential equations with arbitrary terms θ and arbitrary occurrences of x or other variables in θ. As, e.g., in [13,20], however, our calculus assumes that $\dot{x} = \theta$ has a first-order definable flow or approximation. See [2,24] for flow approximation techniques. Extensions of d$\mathcal{L}$ and its calculus to systems of differential equations and higher-order derivatives are accordingly.

The $?\phi$ action is used to define conditions. Its semantics is that of a no-op if ϕ is true in the current state, and that of a failure divergence blocking all further evolution, otherwise. The non-deterministic choice $\alpha \cup \gamma$, sequential composition $\alpha; \gamma$ and non-deterministic repetition α^* of hybrid programs are as usual. They can be combined with $?\phi$ to form more complicated control structures, see [15].

In d$\mathcal{L}$, there is no need to distinguish between discrete and continuous variables or between system parameters and state variables, as they share the same uniform semantics. For instance, $\exists z\, [\dot{z} = -z] z \leq 5$ expresses that there is a choice of the initial value for z (which could be a parameter) such that after all evolutions of z along $\dot{z} = -z$, the outcome of the state variable z will be at most 5. For pragmatic reasons, an informal distinction can improve readability. Formal distinctions of quantified variables and state variables [4] carry over to d$\mathcal{L}$.

2.2 Semantics

Hybrid systems evolve along a piecewise continuous trajectory in n-dimensional space as time passes (see Fig. 1 for a possible evolution with one system variable x over time t). The discontinuities are caused by discrete jumps in the state space while the segments of continuous evolution are governed by differential equations. Concerning semantics of hybrid system models, there is a variety of slightly different formalisations. Since the interplay of discrete change with continuous evolution raises peculiar subtleties, we start with a motivation that highlights the advantages of our choice of semantics for d$\mathcal{L}$.

Motivation. Consider the scenario in Fig. 1. The semantics has to restrict the behaviour of the hybrid system during the continuous evolution phase, e.g., on the interval $[1, 2]$, to respect the differential equation $\dot{x} = f(x)$. Yet, the discrete jump at time 2 will necessarily lead to a discontinuity in the overall system trajectory. Now, an overall system trajectory function g (where $g(t)$ records the value of x at time t) can only assume a *single* value at time 2, say $g(2) = 0.6$. Hence, the evolution of g will only be continuous on the inner interval $(1, 2)$. Still, the evolution along $\dot{x} = f(x)$ has to be constrained to possess a *left*-continuous continuation at time 2 towards a projected value of 1, although this value will never be assumed by g. This complicates the well-posed definition of semantics on the basis of an overall system trajectory. Note that leaving out this condition of left-continuity would lead to a total transition relation with *all* states being reachable, which, of course, would not reflect the proper system behaviour either.

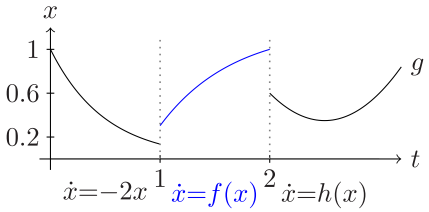


Fig. 1. Discontinuous hybrid trajectory

In contrast to this, the d$\mathcal{L}$ semantics inflates points in time with instantaneous discrete progression by associating an *individual* trajectory for each continuous evolution or instant jump phase. Hence, the trajectories remain continuous within each differential evolution phase, with discontinuities isolated purely in discrete jump transitions. Thereby, the d$\mathcal{L}$ semantics directly traces the succession of values assumed during the hybrid evolution, even if they belong to states which occur without model time passing in between. In addition to the fact that those so-called *super-dense* time effects naturally occur at mode switches between differential evolutions, they are necessary for joint mode switches of several system variables at once, like in $x := 3; y := 5$. We argue that the d$\mathcal{L}$ semantics is much simpler to define than for approaches with a global overall system trajectory as, for example, in [9].

Formal Semantics. The interpretations of d$\mathcal{L}$ consist of states (worlds) that are first-order structures over the reals, with state variables progressing along a sequence of states. A potential behaviour of a hybrid system corresponds to a sequence of states that contain the observable values of system variables during its hybrid evolution. More precisely, the semantics of a single (compound or elementary) system action is captured by the state transitions that are possible using this action. For discrete jumps α this transition relation $\rho(\alpha)$ holds for pairs of states that satisfy the jump constraint. In case of continuous evolutions, the transition relation holds for pairs of states that can be interconnected by a continuous system flow that respects the differential equation, thereby hiding the intermediate flow details from the logic. To retain a manageable logic and calculus, it is important to hide as much as possible of the branching factor of continuous evolution from the logic. Since function and predicate symbols are interpreted as usual for real arithmetic, we omit first-order structures from the notation and focus on states, i.e, assignments of variables with real values.

Definition 2 (State). *A state is a map* $\nu : V \to \mathbb{R}$*; the set of all states is denoted by* $\mathrm{Sta}(V)$*. An* interpretation *is a non-empty set of states that is closed under hybrid program operations (see Def. 4).*

Further, we use $\nu[x \mapsto d]$ to denote the *semantic modification* of a state ν that is identical to ν except for the interpretation of the symbol x, which is $d \in \mathbb{R}$. With the exception of continuous evolution, the semantics, $\rho(\alpha)$, of hybrid program α as a state transition structure in $\mathsf{d}\mathcal{L}$ is as customary in dynamic logic (Def. 4).

Definition 3 (Valuation of terms and formulas). *For terms and formulas, the* valuation $val(\nu, \cdot)$ *with respect to state* ν *is defined as usual for first-order modal logic (e.g. [15]), i.e., using the following definitions for modal operators:*

1. $val(\nu, [\alpha]\phi) = true :\Longleftrightarrow val(\omega, \phi) = true$ *for all* ω *with* $(\nu, \omega) \in \rho(\alpha)$
2. $val(\nu, \langle\alpha\rangle\phi) = true :\Longleftrightarrow val(\omega, \phi) = true$ *for some* ω *with* $(\nu, \omega) \in \rho(\alpha)$

Definition 4 (Semantics of hybrid programs). *The* valuation, $\rho(\alpha)$, of a hybrid program α*, is a* transition relation *on states. It specifies which state* ω *is reachable from a state* ν *by operations of the hybrid system* α *and is defined as:*

1. $(\nu, \omega) \in \rho(x := \theta) :\Longleftrightarrow \omega = \nu[x \mapsto val(\nu, \theta)]$
2. $(\nu, \omega) \in \rho(\dot{x} = \theta \,\&\, \chi) :\Longleftrightarrow$ *there is a function* $f : [0, r] \to \mathrm{Sta}(V)$ *with* $r \geq 0$ *such that* $f(0) = \nu, f(r) = \omega$*, and* $val(f(\zeta), x)$ *is continuous in* ζ *on* $[0, r]$ *and has a derivative of value* $val(f(\zeta), \theta)$ *at each time* $\zeta \in (0, r)$*. For* $y \neq x$ *and* $\zeta \in [0, r]$, $val(f(\zeta), y) = val(\nu, y)$*. Further,* $val(f(\zeta), \chi) = true$ *for each* $\zeta \in (0, r)$*. Systems of differential equations are defined accordingly.*
3. $\rho(?\chi) = \{(\nu, \nu) \;:\; val(\nu, \chi) = true\}$
4. $\rho(\alpha; \gamma) = \rho(\alpha) \circ \rho(\gamma) = \{(\nu, \omega) : (\nu, z) \in \rho(\alpha), (z, \omega) \in \rho(\gamma)$ *for some state* $z\}$
5. $\rho(\alpha \cup \gamma) = \rho(\alpha) \cup \rho(\gamma)$
6. $(\nu, \omega) \in \rho(\alpha^*)$ *iff there are* $n \in \mathbb{N}$ *and* $\nu{=}\nu_0, \ldots, \nu_n{=}\omega$ *with* $(\nu_i, \nu_{i+1}) \in \rho(\alpha)$ *for* $0 \leq i < n$*.*

For the semantics of differential equations, derivatives are well-defined on $(0, r)$ as $\mathrm{Sta}(V)$ is isomorphic to a finite dimensional real space when V is finite.

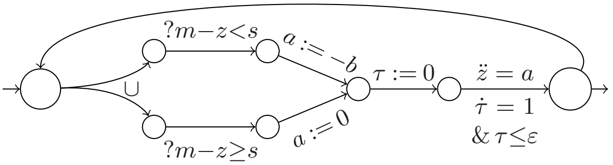


Fig. 2. Transition structure of speed supervision

2.3 Speed Supervision in Train Control

In the European Train Control System (ETCS) [9, 12], trains are only allowed to move within their current movement authority block (MA). When their MA is not extended before reaching its end, trains always have to stop within the MA because there can be open gates or other trains beyond. Here, we identify a single component which is most responsible for the hybrid characteristics of safe driving: speed supervision locally controls the movement of a train such that it always remains within its MA. Depending on the current driving situation, the speed supervision determines a safety envelope s around the train, within which driving is safe, and adjusts its acceleration a in accordance with s (called *correction* in [9]). In the course of this paper, we derive a constraint on s that guarantees safe driving. To simplify the presentation, we assume, as in [9], that the train controller only chooses between braking and keeping speed. Constraints for positive acceleration can be derived accordingly.

Of course, a safe operation of ETCS also depends on other aspects like a disjoint partitioning of the track into MA, appropriate computation of the safe rear end of trains, or proper functioning of gates [9]. But these more static properties are much easier to show when the most important hybrid train dynamics have been captured in a reliable operation of the speed supervision.

We assume that an MA has been granted up to track position m and the train is located at position z, heading with initial speed v towards m. In this situation, d$\mathcal{L}$ can analyse the following safety property of speed supervision:

$$\begin{aligned} \psi &\;\rightarrow\; [(\mathit{corr}; \mathit{drive})^*]\, z \leq m \qquad (1)\\ \text{where } \mathit{corr} &\;\equiv\; (?m - z < s; a := -b) \cup (?m - z \geq s; a := 0)\\ \mathit{drive} &\;\equiv\; \tau := 0; (\dot{z} = v, \dot{v} = a, \dot{\tau} = 1 \,\&\, v \geq 0 \land \tau \leq \varepsilon)\ . \end{aligned}$$

It expresses that a train will *always* remain within its MA m, assuming a constraint ψ for the parameters. In *corr*, the train corrects its acceleration or brakes with force b (as a failsafe recovery manoeuvre [9]) on the basis of the remaining distance $(m-z)$. Then, the train continues moving according to *drive*. There, the position z of the train evolves according to the system $\dot{z} = v, \dot{v} = a$ (i.e., $\ddot{z} = a$). The evolution stops when the speed v drops below zero (or earlier). Simultaneously, clock τ measures the duration of the current *drive* phase before the controllers react to situation changes (we model this to bridge the gap of continuous-time models and discrete-time control design). Clock τ is reset to zero when entering *drive*, constantly evolves along $\dot{\tau} = 1$, and is bound by the invariant region $\tau \leq \varepsilon$. The effect is that a *drive* phase is interrupted for reassessing the driving situation after at most ε seconds, and the *corr*; *drive* loop repeats. The corresponding transition structure $\rho((corr; drive)^*)$ is depicted in Fig. 2.

Instead of manually choosing specific values for the free parameters as in [9, 12], we will use our calculus to synthesise constraints on the relationship of parameters that are required for a safe operation of train control.

3 A Verification Calculus for Differential Logic

In this section, we introduce a sequent calculus for verifying hybrid systems in $\mathsf{d}\mathcal{L}$. With the basic idea being to perform a symbolic evaluation, hybrid programs are successively transformed into logical formulas describing their effects.

For propositional logic, standard rules P1–P9 are listed in Fig. 3. The other rules transform hybrid programs into simpler logical formulas, thereby relating the meaning of programs and formulas. Rules D1–D7 are as in discrete dynamic logic [4, 15]. D8 uses generalised substitutions [4] for handling discrete change. Unlike in uninterpreted first-order logic [15], quantifiers are dealt with using quantifier elimination [8] over the reals (D9–D12) in a way that is compatible with dynamic modalities. D13–D14 handle continuous evolutions given a first-order definable flow y_x for $\dot{x} = \theta$. In combination with D9–D12, they fully encapsulate the handling of differential equations within hybrid systems. D15 is an induction schema with inductive invariant p.

3.1 Rules of the Calculus

A *sequent* is of the form $\Gamma \vdash \Delta$, where Γ and Δ are finite sets of formulas. Its semantics is that of the formula $\bigwedge_{\phi \in \Gamma} \phi \rightarrow \bigvee_{\psi \in \Delta} \psi$. Sequents will be treated as an abbreviation. In the following, an *update* $\mathcal{U}$ simply is a list of discrete assignments of the form $x := \theta$ (see [4] for techniques dealing with a single parallel update rather than a list, which can be combined with our calculus). Rules are applicable anywhere in the sequent, within any context Γ, Δ and update $\langle\mathcal{U}\rangle$:

Definition 5 (Provability, derivability). *A formula ψ is* provable *from a set Φ of formulas, denoted by $\Phi \vdash_{\mathsf{d}\mathcal{L}} \psi$ iff there is a finite set $\Phi_0 \subseteq \Phi$ for which the sequent $\Phi_0 \vdash \psi$ is derivable. In turn, a sequent of the form $\Gamma, \langle\mathcal{U}\rangle\Phi \vdash \langle\mathcal{U}\rangle\Psi, \Delta$ (for some update $\mathcal{U}$, including the empty update, and finite sets Γ, Δ of context formulas) is* derivable *iff there is an instance*

$$\frac{\Phi_1 \vdash \Psi_1 \qquad \ldots \qquad \Phi_n \vdash \Psi_n}{\Phi \vdash \Psi}$$

of a rule schema of the $\mathsf{d}\mathcal{L}$ calculus in Fig. 3 such that for each $1 \leq i \leq n$

$$\Gamma, \langle\mathcal{U}\rangle\Phi_i \vdash \langle\mathcal{U}\rangle\Psi_i, \Delta$$

$$\text{(P1)}\ \frac{\vdash \phi}{\neg\phi \vdash} \qquad \text{(P4)}\ \frac{\phi, \psi \vdash}{\phi \land \psi \vdash} \qquad \text{(P7)}\ \frac{\phi \vdash \qquad \psi \vdash}{\phi \lor \psi \vdash}$$

$$\text{(P2)}\ \frac{\phi \vdash}{\vdash \neg\phi} \qquad \text{(P5)}\ \frac{\vdash \phi \qquad \vdash \psi}{\vdash \phi \land \psi} \qquad \text{(P8)}\ \frac{\vdash \phi, \psi}{\vdash \phi \lor \psi}$$

$$\text{(P3)}\ \frac{\phi \vdash \psi}{\vdash \phi \to \psi} \qquad \text{(P6)}\ \frac{\vdash \phi \qquad \psi \vdash}{\phi \to \psi \vdash} \qquad \text{(P9)}\ \frac{}{\phi \vdash \phi}$$

$$\text{(D1)}\ \frac{\phi \land \psi}{\langle ?\phi \rangle \psi} \qquad \text{(D5)}\ \frac{\phi \lor \langle \alpha; \alpha^* \rangle \phi}{\langle \alpha^* \rangle \phi} \qquad \text{(D9)}\ \frac{\mathsf{qelim}(\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma \vdash \Delta, \exists x\, \phi}$$

$$\text{(D2)}\ \frac{\phi \to \psi}{[?\phi]\psi} \qquad \text{(D6)}\ \frac{\phi \land [\alpha; \alpha^*]\phi}{[\alpha^*]\phi} \qquad \text{(D10)}\ \frac{\mathsf{qelim}(\forall x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma, \exists x\, \phi \vdash \Delta}$$

$$\text{(D3)}\ \frac{\langle \alpha \rangle \phi \lor \langle \gamma \rangle \phi}{\langle \alpha \cup \gamma \rangle \phi} \qquad \text{(D7)}\ \frac{\langle\![\alpha]\!\rangle \langle\![\gamma]\!\rangle \phi}{\langle\![\alpha; \gamma]\!\rangle \phi} \qquad \text{(D11)}\ \frac{\mathsf{qelim}(\forall x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma \vdash \Delta, \forall x\, \phi}$$

$$\text{(D4)}\ \frac{[\alpha]\phi \land [\gamma]\phi}{[\alpha \cup \gamma]\phi} \qquad \text{(D8)}\ \frac{\phi_x^\theta}{\langle\![x := \theta]\!\rangle \phi} \qquad \text{(D12)}\ \frac{\mathsf{qelim}(\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma, \forall x\, \phi \vdash \Delta}$$

$$\text{(D13)}\ \frac{\exists t {\geq} 0\, (\bar\chi \land \langle x := y_x(t) \rangle \phi)}{\langle \dot x = \theta \,\&\, \chi \rangle \phi} \qquad \text{(D14)}\ \frac{\forall t {\geq} 0\, (\bar\chi \to [x := y_x(t)] \phi)}{[\dot x = \theta \,\&\, \chi] \phi}$$

$$\text{(D15)}\ \frac{\vdash p \qquad \vdash [\alpha^*](p \to [\alpha]p)}{\vdash [\alpha^*]p}$$

Rule D8 is only applicable if the substitution of x by θ in ϕ_x^θ introduces no new bindings. In D13–D14, t and $\tilde t$ are fresh variables, and y_v is the solution of the initial value problem $(\dot x = \theta, x(0) = v)$. Additionally, $\bar\chi$ is an abbreviation for $\forall 0{<}\tilde t{<}t\, \langle x := y_x(\tilde t) \rangle \chi$; it simplifies to *true* if χ equals *true*. In D9–D12, x does not occur in Γ, Δ. Further, the $\Gamma_i \vdash \Delta_i$ are obtained from the resulting sub-goals of a side deduction, see ($\star$) in Fig. 4. The side deduction is started from the goal $\Gamma \vdash \Delta, \phi$ at the bottom (or $\Gamma, \phi \vdash \Delta$ for D10 and D12). In the resulting sub-goals $\Gamma_i \vdash \Delta_i$, variable x is assumed to occur in first-order formulas only, as quantifier elimination (qelim) is then applicable.

Fig. 3. Rule schemata of the d$\mathcal{L}$ verification calculus

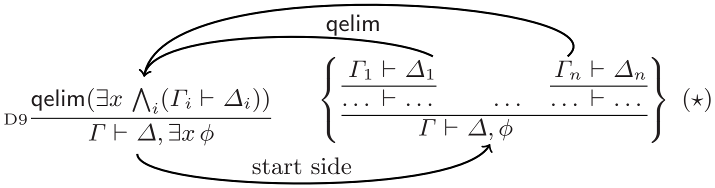

qelim
D9
$\mathsf{qelim}(\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i))$
$\Gamma \vdash \Delta, \exists x\, \phi$
$\Gamma_1 \vdash \Delta_1$
$\dots \vdash \dots$
$\dots$
$\Gamma_n \vdash \Delta_n$
$\dots \vdash \dots$
$\Gamma \vdash \Delta, \phi$
($\star$)
start side

Fig. 4. Side deduction for quantifier elimination rules

is derivable. Moreover, the symmetric schemata D1–D14 can be applied on either side of the sequent (again in context Γ, Δ and update $\langle \mathcal{U} \rangle$). In D7 and D8, the schematic modality $\langle\![\cdot]\!\rangle$ can further be instantiated with both $[\cdot]$ and $\langle \cdot \rangle$. The same modality instance has to be chosen within a single schema instantiation, though.

As usual in sequent calculus—although the direction of entailment is from premisses (above rule bar) to conclusion (below)—the order of reasoning is *goal-directed*: Rules are applied in tableau-style, that is, starting from the desired conclusion at the bottom (goal) to the premisses (sub-goals). In the sequel we illustrate the new $\mathsf{d}\mathcal{L}$ rules.

Discrete jumps. For handling discrete change, rule D8 uses generalised substitutions [4]. The result of applying to ϕ the substitution that replaces x by θ is defined as usual [15]; it is denoted by ϕ_x^θ. Rule D8 is not applicable when the substitution introduces new bindings. For this, the definition of a "bound occurrence of a variable y" is amended to include the scope of $y := \theta$ and $\dot{y} = \theta$, because both change the value of y.

Continuous evolution. D13–D14 require solving a symbolic initial value problem. We assume that the differential equations have first-order definable unique flows. See [2, 24] for first-order approximation techniques of more general flows.

Real arithmetic. Rules D9–D12 constitute a purely modular interface to mathematical reasoning. They can use any theory that admits quantifier elimination and has a decidable ground theory (e.g., [8]):

Definition 6 (Quantifier elimination). *A first-order theory admits* quantifier elimination *if to each formula ϕ, a quantifier-free formula* $\mathsf{qelim}(\phi)$ *can be effectively associated that is equivalent (i.e., $\phi \leftrightarrow \mathsf{qelim}(\phi)$ is valid) and has no other free variables. The operation* qelim *is further assumed to evaluate ground formulas (i.e., without variables), yielding a decision procedure for this theory.*

Integrating quantifier elimination to deal with statements about real quantities is quite challenging in the presence of modalities that influence the value of variables and terms. Even more so, the effect of a modality depends on the solutions of the differential equations contained therein. For instance, it is hard to know in advance, which first-order constraints need to be solved by qelim in $\exists x\, [\ddot{x} = -b; x := 2x]x \geq 5$. To find out, the way how x evolves from $\exists x$ to $x \geq 5$ along the system dynamics needs to be taken into account. Hence, our calculus first unveils the first-order constraints on x before applying qelim. To achieve this in a concise way, we use side deductions.

The effect of a side deduction is as follows. First, the $\mathsf{d}\mathcal{L}$ calculus discovers all relevant first-order constraints from modal formulas using a side deduction in $\mathsf{d}\mathcal{L}$. Secondly, these constraints are equivalently reduced using qelim and the main proof continues. For instance, an application of D9 to a sequent $\Gamma \vdash \Delta, \exists x\, \phi$ starts a side deduction (marked ($\star$) in Fig. 4) with the goal $\Gamma \vdash \Delta, \phi$ at the bottom. This side deduction is carried out in the $\mathsf{d}\mathcal{L}$ calculus until x no longer occurs within modal formulas of the remaining open branches $\Gamma_i \vdash \Delta_i$ of ($\star$). Once all occurrences of x are in first-order formulas, the resulting sub-goals $\Gamma_i \vdash \Delta_i$ of ($\star$) are copied back to the main proof and qelim is applied (which determines the resulting sub-goal of rule D9 on the upper left side of Fig. 4). The remaining modal formulas not containing x can be considered as atoms for this purpose as they do not impose constraints on x.

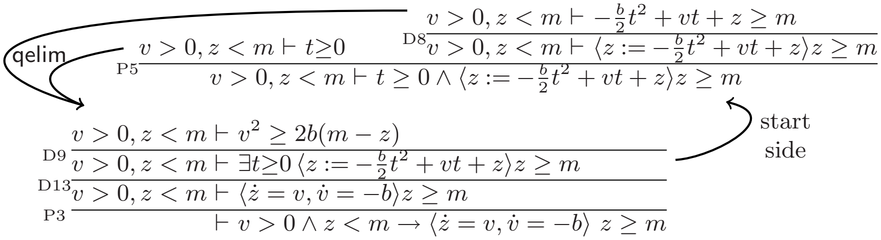

$v > 0, z < m \vdash -\frac{b}{2}t^2 + vt + z \geq m$
D8
$v > 0, z < m \vdash \langle z := -\frac{b}{2}t^2 + vt + z\rangle z \geq m$
qelim
$v > 0, z < m \vdash t \geq 0$
P5
$v > 0, z < m \vdash t \geq 0 \land \langle z := -\frac{b}{2}t^2 + vt + z\rangle z \geq m$
start side
$v > 0, z < m \vdash v^2 \geq 2b(m - z)$
D9
$v > 0, z < m \vdash \exists t{\geq}0\, \langle z := -\frac{b}{2}t^2 + vt + z\rangle z \geq m$
D13
$v > 0, z < m \vdash \langle \dot{z} = v, \dot{v} = -b\rangle z \geq m$
P3
$\vdash v > 0 \land z < m \rightarrow \langle \dot{z} = v, \dot{v} = -b\rangle\ z \geq m$

Fig. 5. Analyse MA-violation in braking mode using side deductions

For implementations in a theorem prover, a careful analysis shows that side deductions can also be performed within the original proof, but the corresponding calculus rules, which keep track of the (lost) quantifier nesting, are quite technical and side deductions lead to a cleaner proof structure. Observe that reintegrating the open branches $\Gamma_i \vdash \Delta_i$ into the main proof corresponds to discharging multiple sub-goals simultaneously. Modifying tableau procedures to remove multiple open branches at once is not difficult. It works similarly to simultaneous closing substitutions in all branches of free variable tableaux, except that quantifier elimination can produce more than one instantiation.

3.2 Modular Combination by Side Deduction

To illustrate how our calculus combines arithmetic with dynamic reasoning using side deductions, we give an example. Because we will need a similar result when verifying speed supervision, we consider braking of trains. The deduction in Fig. 5 can be used to analyse whether a train could violate its MA although it brakes. As the prover will discover, the answer depends on the initial velocity v.

Rules D9 and D13 are implemented using Mathematica. Applying D9 in the main proof triggers a side deduction. The conjunction of the open proof goals in the side deduction can be handled by quantifier elimination and simplification in Mathematica, yielding the resulting premiss for D9 in the main proof:

$$\mathsf{qelim}\big(\exists t\, \big((v > 0 \land z < m \rightarrow t \geq 0) \land (v > 0 \land z < m \rightarrow -\frac{b}{2}t^2{+}vt{+}z \geq m)\big)\big)$$
$$\equiv\ v > 0 \land z < m\ \rightarrow\ v^2 \geq 2b(m - z)\ .$$

The open branch of the main proof reveals the speed limit and can be used to synthesise a corresponding parametric constraint. When $v^2 \geq 2b(m - z)$ holds initially, then the MA will be violated despite braking. Similarly, $v^2 \leq 2b(m - z)$ guarantees that the MA can be respected by appropriate braking.

3.3 Soundness and Incompleteness

In this section we prove that verification with the $\mathsf{d}\mathcal{L}$ calculus always produces correct results about system safety, i.e., the $\mathsf{d}\mathcal{L}$ calculus is sound.

Theorem 1 (Soundness). *The* d$\mathcal{L}$ *calculus is* sound*, i.e., derivable formulas are* valid *(true in all states of all interpretations).*

This theorem is a direct consequence of an even stronger result of soundness localised with respect to a single state (instead of requiring the premiss to be true in all states). The primary challenges within this proof are continuous evolutions and the interaction of quantifier elimination with sequent calculus.

Proposition 1 (Local soundness). *All* d$\mathcal{L}$ *rules in Fig. 3 are* locally sound*: for all states ν, the conclusion is true in ν when all premisses are true in ν.*

Proof. Most rules can be proven as in [4]. The special cases for d$\mathcal{L}$ are:

- Rule D9 is locally sound: Let ν be a state in which the premiss is true, i.e.,

  $$\nu \models \mathsf{qelim}(\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i)) \ .$$

  We have to show that the conclusion is true in this state. Using that quantifier elimination yields an equivalence, we see that ν also satisfies $\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i)$ prior to the quantifier elimination. Hence, for some $d \in \mathbb{R}$ we obtain:

  $$\nu[x \mapsto d] \models \bigwedge_i (\Gamma_i \vdash \Delta_i) \ .$$

  As $(\star)$ in Fig. 4 is inductively shown to be locally sound, we can conclude that $\nu[x \mapsto d] \models (\Gamma \vdash \Delta, \phi)$. Therefore, $\nu \models \exists x\,(\Gamma \vdash \Delta, \phi)$. Now the conjecture can be obtained using standard reasoning with quantifiers and the absence of x in Γ, Δ by rewriting the conclusion with local equivalences:

  $$\exists x\,(\Gamma \vdash \Delta, \phi) \ \equiv\ \exists x\,(\neg\Gamma \lor \Delta \lor \phi) \ \equiv\ \neg\Gamma \lor \Delta \lor \exists x\,\phi \ \equiv\ \Gamma \vdash \Delta, \exists x\,\phi \tag{2}$$

  The soundness proof for D11 is similar since (2) holds for any quantifier. The proofs of D10 and D12 can be derived using duality of quantifiers.
- Rule D13 is locally sound: Assuming the premiss is true in some state ν, we have to show that there is a state ω with $\omega \models \phi$ such that $(\nu, \omega) \in \rho(\dot{x} = \theta)$. By premise, there is a real value $e \geq 0$ such that when we abbreviate $\nu[t \mapsto e]$ by $\tilde{\nu}$, we have $\tilde{\nu} \models \langle x := y_x(t)\rangle\phi$. Let ω_e be such that $(\tilde{\nu}, \omega_e) \in \rho(x := y_x(t))$, thus $\omega_e \models \phi$. Then, it only remains to show $(\tilde{\nu}, \omega_e) \in \rho(\dot{x} = \theta)$. This, in turn, is shown using the function $e \mapsto \omega_e$, which yields continuity and a solution of the initial value problem by the corresponding properties of y_x. Since $\rho(\dot{x} = \theta)$ and ϕ do not depend on the fresh variable t, the same reasoning holds for ν in place of $\tilde{\nu}$. The invariant χ can be shown accordingly.

Now we show that unlike first-order *real* arithmetic, d$\mathcal{L}$ is undecidable. We show that both unbounded repetition in the discrete fragment and unbounded evolution in the continuous fragment cause incompleteness and undecidability.

Theorem 2 (Incompleteness). *Both the discrete fragment and the continuous fragment of* d$\mathcal{L}$ *are inherently* incomplete*, i.e., there is no sound and complete effective calculus. Hence, valid* d$\mathcal{L}$ *formulas are not always provable.*

$$\text{(G1)}\ \frac{\phi \vdash \psi}{\langle\!\![\alpha]\!\!\rangle\phi \vdash \langle\!\![\alpha]\!\!\rangle\psi} \qquad\qquad \text{(G2)}\ \frac{\Gamma \vdash \langle\!\![\mathcal{U}]\!\!\rangle p, \Delta \qquad p \vdash [\alpha]p \qquad p \vdash \phi}{\Gamma \vdash \langle\!\![\mathcal{U}]\!\!\rangle[\alpha^*]\phi, \Delta}$$

Fig. 6. Global and derived d$\mathcal{L}$ rules

Proof. We prove that natural numbers are definable amongst the real numbers of d$\mathcal{L}$ interpretations in both fragments. Then these fragments extend first-order *integer* arithmetic such that the incompleteness theorem of Gödel applies. Natural numbers are definable in the discrete fragment using repetitive additions. In the continuous fragment without $\{^*, :=\}$, natural numbers are definable as:

$$nat(n) \;\leftrightarrow\; \exists s\, \exists c\, (s = 0 \land c = 1 \land \langle \dot{s} = c, \dot{c} = -s, \dot{\tau} = 1\rangle(s = 0 \land \tau = n)) \;\; .$$

These ODEs have *sin* and *cos* as unique solutions for s and c, respectively. Their zeros, detected by τ, characterise an isomorphic copy of natural numbers, scaled by π. The initial values for s and c prevent the trivial solution identical to 0.

4 Verifying Speed Supervision in Train Control

Finding Inductive Candidates. We want to prove the safety statement (1) of Section 2.3. Using parametric extraction techniques, we identify both the requirement ψ for safe driving and the induction hypothesis p that is required for the proof. An unwinding of the loop in (1) by D6 can be used to extract a candidate for a parametric inductive hypothesis (similar to the proof in Section 3.2). It expresses that there is sufficient braking distance $(m - z)$ at current speed v:

$$p \;\equiv\; v^2 \leq 2b(m - z) \land b > 0 \land \varepsilon > 0 \;\; .$$

Inductive Verification. The generalisation rule G1 in Fig. 6 can be used to derive the variant G2 of the induction rule D15: With G1, G2 can be derived from D15 by discharging $[\alpha^*]$ in the premiss of D15 and strengthening ϕ to p.

Applying G2 to (1), the premiss $p \vdash z \leq m$ holds as $0 \leq v^2 \leq 2b(m - z)$ and $b > 0$. The induction start $\psi \vdash p$ of G2 will be examined after the full proof has been given, since we want to identify the prerequisite ψ for safe driving by proof analysis. For proving the induction step $p \vdash [corr; drive]p$, we remove condition $m - z < s$ from *corr*, because it is not used in the proof (as braking remains safe with respect to $z \leq m$). Here, we abbreviate the side deductions of D11 as they do not branch but only apply P3,D8. The induction step of G2 can be proven in d$\mathcal{L}$ (D11 and D14 are implemented in Mathematica):

$$\text{D7}\ \dfrac{\text{D4,P5}\ \dfrac{\dfrac{\cdots}{p \vdash [a := -b][drive]p} \qquad \text{D2,P3}\ \dfrac{\dfrac{\cdots}{p, m{-}z{\geq}s \vdash [a := 0][drive]p}}{p \vdash [?m{-}z{\geq}s; a := 0][drive]p}}{p \vdash [corr][drive]p}}{p \vdash [corr; drive]p}$$

Here, the invariant evolution conditions are convex, hence $\bar{\chi}$ can be simplified to $\langle x := y_x(t)\rangle\chi$ to save space. Further, we leave out conditions which are unnecessary for closing the proof: In the left branch, the constrained evolution of τ is irrelevant and will be left out. The left branch closes as follows (marked as $*$):

$$\begin{array}{rl}
 & \qquad * \\ \cline{2-2}
\text{D11,P3,D8} & p \vdash \forall t{\ge}0\,([v := -bt+v]v \ge 0 \to [z := -\frac{b}{2}t^2 + vt + z; v := -bt+v]p) \\ \cline{2-2}
\text{D14} & p \vdash [\dot{z} = v, \dot{v} = -b \,\&\, v \ge 0]p \\ \cline{2-2}
\text{D8} & p \vdash [a := -b][drive]p
\end{array}$$

The right branch does not need condition $v \ge 0$, because v does not decrease:

$$\begin{array}{rl}
 & \qquad\qquad\quad \dots \\ \cline{2-2}
 & p, m{-}z{\ge}s \vdash v^2 \le 2b(m - \varepsilon v - z) \\ \cline{2-2}
\text{D11,P3,D8} & p, m{-}z{\ge}s \vdash \forall t{\ge}0\,([\tau := t]\tau \le \varepsilon \to [z := vt + z]p) \\ \cline{2-2}
\text{D8} & p, m{-}z{\ge}s \vdash [\tau := 0]\forall t{\ge}0\,([\tau := t + \tau]\tau \le \varepsilon \to [z := vt + z]p) \\ \cline{2-2}
\text{D14} & p, m{-}z{\ge}s \vdash [\tau := 0][\dot{z} = v, \dot{v} = 0, \dot{\tau} = 1 \,\&\, \tau \le \varepsilon]p \\ \cline{2-2}
\text{D8} & p, m{-}z{\ge}s \vdash [a := 0][\tau := 0][\dot{z} = v, \dot{v} = a, \dot{\tau} = 1 \,\&\, \tau \le \varepsilon]p \\ \cline{2-2}
\text{D7} & p, m{-}z{\ge}s \vdash [a := 0][drive]p
\end{array}$$

Augmenting Inductive Candidates. The right branch only closes when p guarantees the succedent $v^2 \le 2b(m - \varepsilon v - z)$, i.e., that there will still be sufficient braking distance even after keeping the speed for up to ε seconds. As $m - z \ge s$, this succedent is implied by $v^2 \le 2b(s - \varepsilon v)$, which can be discovered automatically by quantifier elimination. In fact, using $p \wedge v^2 \le 2b(s - \varepsilon v)$ in place of p makes the argument inductive, and the whole proof of the safety statement (1) closes when the same formula is chosen for ψ. Here, no conjunct of ψ can be removed without leaving the proof open due to a counterexample.

From $v^2 \le 2b(s - \varepsilon v)$, we can also derive $s \ge \varepsilon v + \frac{v^2}{2b}$ as an equivalent yet constructive constraint. From the above proof, we can further conclude that speed supervision remains safe even when s is chosen *adaptively* in accordance with this constraint at the beginning of *corr* in response to speed changes. This permits safe behaviour as complex as that in Fig. 7. Similar correctness constraints can be derived when the train is allowed to increase its speed if $m - z \ge s$.

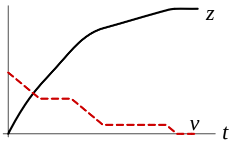


Fig. 7. Correcting

In this example, we can see the effect of the d$\mathcal{L}$ calculus. It takes a specification of a hybrid system and successively identifies the arithmetic constraints which need to be investigated for proving correctness. These constraints can then be handled in a purely modular way by D9-D12 and side deductions.

5 Conclusions and Future Work

We have introduced a first-order dynamic logic with interacting discrete jumps and continuous evolutions along differential equations. For this differential logic, d$\mathcal{L}$, we have presented a calculus for verifying parametric hybrid systems.

Our sequent calculus for $\mathsf{d}\mathcal{L}$ is based on a classical calculus for discrete dynamic logic [15]. In order to handle continuous evolution, we combine quantifier elimination with the calculus in a modular and constructive way. Our calculus handles first-order definable flows for differential equations. It combines non-invasively with deductive verification systems for dynamic logic. Further, it has a modular interface to combine arithmetic with dynamic reasoning in the presence of state change. We demonstrate that our calculus can verify safety in a parametric train control scenario. Meanwhile, this case study has also been verified in an interactive theorem prover based on the KeY system [3].

We currently extend our partial implementation of the $\mathsf{d}\mathcal{L}$ calculus. Moreover, dynamic logic supports reasoning about dynamic reconfiguration of system structure [4], which we want to extend to hybrid systems. Finally, our future ambition is to analyse the quotient of reasoning about hybrid systems modulo differential equation solving and inductive first-order system invariants.

Acknowledgements. I would like to thank the anonymous referees for their insightful comments and Ernst-Rüdiger Olderog and his group for their remarks.

References

1. Alur, R., Henzinger, T.A., Ho, P.-H.: Automatic symbolic verification of embedded systems. IEEE Trans. Software Eng. 22(3), 181–201 (1996)
2. Anai, H., Weispfenning, V.: Reach set computations using real quantifier elimination. In: Benedetto, M.D.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 63–76. Springer, Heidelberg (2001)
3. Beckert, B., Hähnle, R., Schmitt, P.H. (eds.): Verification of Object-Oriented Software. LNCS (LNAI), vol. 4334. Springer, Heidelberg (2007)
4. Beckert, B., Platzer, A.: Dynamic logic with non-rigid functions: A basis for object-oriented program verification. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 266–280. Springer, Heidelberg (2006)
5. Bemporad, A., Bicchi, A., Buttazzo, G. (eds.): Hybrid Systems: Computation and Control. 10th International Conference, HSCC 2007, Pisa, Italy. LNCS, vol. 4416. Springer, Heidelberg (2007)
6. Boulton, R.J., Hardy, R., Martin, U.: A Hoare logic for single-input single-output continuous-time control systems. In: Maler, O., Pnueli, A. (eds.) HSCC 2003. LNCS, vol. 2623, pp. 113–125. Springer, Heidelberg (2003)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
8. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. 12(3), 299–328 (1991)
9. Damm, W., Hungar, H., Olderog, E.-R.: On the verification of cooperating traffic agents. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2003. LNCS, vol. 3188, pp. 77–110. Springer, Heidelberg (2003)
10. Davoren, J.M.: On hybrid systems and the modal μ-calculus. In: Antsaklis, P.J., Kohn, W., Lemmon, M.D., Nerode, A., Sastry, S.S. (eds.) Hybrid Systems. LNCS, vol. 1567, pp. 38–69. Springer, Heidelberg (1997)
11. Davoren, J.M., Nerode, A.: Logics for hybrid systems. Proceedings of the IEEE 88(7), 985–1010 (2000)

12. Faber, J., Meyer, R.: Model checking data-dependent real-time properties of the European Train Control System. In: FMCAD, pp. 76–77. IEEE Computer Society, Washington (2006)
13. Fränzle, M.: Analysis of hybrid systems. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 126–140. Springer, Heidelberg (1999)
14. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 258–273. Springer, Heidelberg (2005)
15. Harel, D., Kozen, D., Tiuryn, J.: Dynamic logic. MIT Press, Cambridge (2000)
16. Henzinger, T.A.: The theory of hybrid automata. In: LICS, pp. 278–292. IEEE Computer Society, Washington (1996)
17. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. In: LICS, pp. 394–406. IEEE Computer Society, Washington (1992)
18. Hutter, D., Langenstein, B., Sengler, C., Siekmann, J.H., Stephan, W., Wolpers, A.: Deduction in the verification support environment (VSE). In: Gaudel, M.-C., Woodcock, J. (eds.) FME 1996. LNCS, vol. 1051, Springer, Heidelberg (1996)
19. Lafferriere, G., Pappas, G.J., Yovine, S.: A new class of decidable hybrid systems. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, pp. 137–151. Springer, Heidelberg (1999)
20. Piazza, C., Antoniotti, M., Mysore, V., Policriti, A., Winkler, F., Mishra, B.: Algorithmic algebraic model checking I. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 5–19. Springer, Heidelberg (2005)
21. Platzer, A.: Differential logic for reasoning about hybrid systems. In: Bemporad et al. [5] p. 746–749
22. Platzer, A.: A temporal dynamic logic for verifying hybrid system invariants. In: Artemov, S., Nerode, A. (eds.) Logical Foundations of Computer Science, International Symposium, LFCS 2007, New York, USA. LNCS, vol. 4514, pp. 457–471. Springer, Heidelberg (2007)
23. Platzer, A.: Towards a hybrid dynamic logic for hybrid dynamic systems. In: Blackburn, P., Bolander, T., Braüner, T., de Paiva, V., Villadsen, J. (eds.), Proc. LICS International Workshop on Hybrid Logic, 2006, Seattle, ENTCS (2007)
24. Platzer, A., Clarke, E.M.: The image computation problem in hybrid systems model checking. In: Bemporad et al. [5] p. 473–486
25. Rönkkö, M., Ravn, A.P., Sere, K.: Hybrid action systems. Theor. Comput. Sci. 290(1), 937–973 (2003)
26. Rounds, W.C.: A spatial logic for the hybrid π-calculus. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 508–522. Springer, Heidelberg (2004)
27. Zhou, C., Ravn, A.P., Hansen, M.R.: An extended duration calculus for hybrid real-time systems. In: Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.) Hybrid Systems. LNCS, vol. 736, pp. 36–59. Springer, Heidelberg (1992)

Improvements to the Tableau Prover PITP

Alessandro Avellone, Guido Fiorino, and Ugo Moscato

Dipartimento di Metodi Quantitativi per le Scienze Economiche e Aziendali,
Università Milano-Bicocca, Piazza dell'Ateneo Nuovo, 1, 20126 Milano, Italy
{alessandro.avellone,guido.fiorino,ugo.moscato}@unimib.it

Abstract. In this paper we discuss the new version of PITP, a procedure to decide propositional intuitionistic logic, which turns out at the moment to be the best propositional prover on ILTP. The changes in the strategy and implementation make the new version of PITP faster and capable of deciding more formulas than the previous one. We give a short account both of the old optimizations and the changes in the strategy with respect to the previous version. We use ILTP library and random generated formulas to compare the implementation described in this paper to the other provers (including our old version of PITP).

1 Introduction

In this article we describe the optimizations and the main features of PITPv2.1, a decision procedure based on a tableau calculus for propositional intuitionistic logic. While some of them have already been described in [4] (we refer to the implementation as PITPv1.0) and [6] (we refer to the implementation as PITPv2.0), two optimizations are new and allow us to have a new implementation outperforming the old version (we refer to this version as PITPv2.1 that can be downloaded from `http://www.dimequant.unimib.it/PITP/index.html`).

Our interest in developing an efficient decision procedure and a theorem prover for propositional intuitionistic logic lies in the applications that such a logic and intermediate logics have in formal software/hardware verification and program synthesis ([7,8,14]). As an example, we quote [5,1] where a version of PITP is used as the kernel both to decide an extension of propositional intuitionistic logic with constructive negation and to compute exact stabilization bounds of combinatorial circuits.

We present some experimental results comparing PITPv2.1 and PITPv2.0. We perform our comparisons both on random generated formulas and on the propositional part of ILTP v1.1.2 benchmark library ([15]). Of 274 propositional benchmarks contained in ILTP v1.1.2, PITPv2.1 decides 238 formulas within the time limit of ten minutes; we point out that PITPv2.1 outperforms PITPv1.0, that decides 215 formulas and PITPv2.0, that decides 234 formulas (see ILTP and its archives [15]).

2 The Calculus and the Optimizations: An Overview

PITPv2.1 is based on our tableau calculus. Full details on the calculus can be found in [9,3].

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 233–237, 2007.

The basic strategy employed by PITP consists in dividing the rules into six groups according to their behaviour with respect to branching (rules with two sets of formulas in the conclusion) and backtracking (rules that are non-invertible). This strategy is well-known and is our starting point. PITPv2.1 uses some optimization techniques to narrow the search space and the number of nodes of a tableau proof. In the following we provide a brief account.

We have adapted Simplification to intuitionistic logic, a technique described in [13] for classical and modal logics. As classical and intuitionistic logic are semantically different, in some cases we cannot apply Simplification (see [6]). However, it is the most effective among our optimization techniques and makes the difference in the performances between PITPv1.0 and PITPv2.0. In particular, formulas SYJ202+1.010, SYJ207+1.005, SYJ208+1.009 to SYJ208+1.020 and SYJ212+1.012 to SYJ212+1.016 of ILTP have been decided by PITPv2.0. In order to make this technique effective, data structure has to be carefully designed (we give an account in Section 3).

There are cases where the proof for a formula can be rewritten (by using a permutation on propositional variables) and used as a proof for another formula. This can happen when the formulas at hand have the same structure as in $\{\mathbf{T}(((P0 \rightarrow (P1 \vee P2)) \rightarrow (P1 \vee P2))), \mathbf{T}(((P2 \rightarrow (P1 \vee P0)) \rightarrow (P1 \vee P0))), \mathbf{T}(((P1 \rightarrow (P2 \vee P0)) \rightarrow (P2 \vee P0))), \mathbf{F}P1, \mathbf{F}P2, \mathbf{F}P0\}$. The proof that $S = \{\mathbf{T}(((P2 \rightarrow (P1 \vee P0)) \rightarrow (P1 \vee P0))), \mathbf{T}(((P1 \rightarrow (P2 \vee P0)) \rightarrow (P2 \vee P0))), \mathbf{T}(P0), \mathbf{F}P1, \mathbf{F}P2)\}$ holds in a Kripke model can be rewritten by substituting respectively $P2$ with $P0$ and vice-versa and becomes a proof that $S' = \{\mathbf{T}(((P0 \rightarrow (P1 \vee P2)) \rightarrow (P1 \vee P2))), \mathbf{T}(((P1 \rightarrow (P2 \vee P0)) \rightarrow (P2 \vee P0))), \mathbf{T}(P2), \mathbf{F}P1, \mathbf{F}P0\}$ has a Kripke model. This is possible because there is a permutation τ on the propositional variables occurring in the set such that when we substitute the propositional variables occurring in S' by using τ we get S. PITPv2.0 and PITPv2.1 do not try all the possible permutations but consider two target formulas having the same structure and try to build a single permutation for the two target formulas.

The semantics of the formulas in which only conjunctions and disjunctions occur coincides with their semantics in classical logic. To bound branching on such formulas we exploit *Regularity* [10], a well-known technique applied in classical theorem proving. In a few words, such formulas are not expanded if they are satisfied by the canonical model defined by the formulas at hand. It turns out that regularity allows us to improve the performances on the class of formulas SYJ202+1 in PITPv1.0 (see [4]).

2.1 Two More Optimizations

PITPv2.0 misses two optimizations that in certain circumstances allow the shrinking of the search space and the width of the proofs. The first technique avoids backtracking and is based on the following consideration. When $\mathbf{F} \rightarrow$, $\mathbf{F}\neg$ or $\mathbf{T} \rightarrow\rightarrow$ are applied and $S = S_c$, then the application of the rule does not require backtracking because from satisfiability of one of the sets in the conclusion we can deduce the realizability of the set in the premise (we emphasize that for sequent calculi with one succedent this idea is trivial). By adding this strategy, PITPv2.1 is able to prove more formulas than PITPv2.0 both on the ILTP library and on randomly generated formulas. In particular, it turns out that PITPv2.1 decides formulas SYJ212+1.017 to SYJ212+1.020, a class of formulas already decided by other provers such as ft and STRIP [16,12]. In the following tables comparisons with the propositional provers included in the ILTP library are provided.

Finally, we have introduced Semantic Branching Technique [11], a technique widely used and derived from DLP procedure. In our case it consists in adding to the calculus the rules

$$\frac{S, \mathbf{F}(p \wedge B)}{S, \mathbf{F}p | S, \mathbf{T}p, \mathbf{F}B}\mathbf{F}\wedge{-prop_1} \qquad \frac{S, \mathbf{F}(A \wedge p)}{S, \mathbf{F}A, \mathbf{T}p | S, \mathbf{F}p}\mathbf{F}\wedge{-prop_2}$$

where p is a propositional variable. Thus the rules introduce "positive" information in the set. Our experiments show that with these rules we never waste time. On the other hand by extending this idea to $\mathbf{T}\wedge$-formulas or generalizing to non-atomic formulas sometimes reduces performances. Experiments show that we can have a decrease in performances if we use it exactly as in classical logic.

3 Data Structure Modifications and Other Implementation Issues

The code of PITP is written in C++; to represent formulas we use a goedelization and we handle them in two layers: in the bottom layer we implement the formula according to its structure and in the top layer we implement sets of signed formulas grouping them according to the strategy described in [4]. In each layer we use many data structures to take into account the optimizations described above. For instance, to implement Simplification we store the number of subformulas of a formula and their parent formulas. Moreover, to implement

Table 1. Result on ILTP v1.1.2 formulas

	ft Prolog	ft C	LJT	STRIP	PITPv2.0	PITPv2.1
solved	188	199	175	202	234	238
(%)	68.6	72.6	63.9	73.7	85.4	86.9
proved	104	106	108	119	129	128
refuted	84	93	67	83	105	110
solved after:						
0-1s	173	185	166	178	207	215
1-10s	5	6	4	11	13	15
10-100s	6	7	2	11	7	6
100-600s	4	1	3	2	7	2
(>600s)	86	75	47	43	39	35
errors	0	0	52	29	1	1

branching and backtracking we use a stack and a data structure to treat the signed formulas contained in every set of signed formulas of a path of the proof tree.

4 Concluding Remarks

Let us conclude this short description of our work by observing that formulas of ILTP have been very important for stimulating us in the search for improvements and optimization; formulas of the ILTP library suggest that classical techniques are effective in the intuitionistic framework too. Of 274 formulas of ILTP, 33 formulas haven't been solved yet by any prover. We think that those formulas will require great efforts both from a theoretical and practical point of view. Also, the growth of the performances of PITP (on January 2006 its first version decided only 171 formulas) proves the fact that it hasn't been difficult to incorporate new sound and complete strategies and this could be due to code modularity and/or the use of genuine tableaux calculi. For instance, we were wondering whether the use of contraction free sequent calculi could have the same readiness in incorporating new strategies. For the future, we are waiting for a new version of ILTP including new formulas for testing PITPv2.1 and we plan to incorporate in our parallel version IPTP [2] the new strategies of PITPv2.1 and run IPTP on a HP Superdome 64 CPUs machine.

Table 2. Provable ILTP v1.1.2 formulas solved by classes

	SYJ202+1 provable	SYJ205+1 provable	SYJ206+1 provable
ft Prolog	07 (516.55)	08 (60.26)	10 (144.5)
ft C	07 (76.3)	09 (85.84)	11 (481.98)
LJT	02 (0.09)	20 (0.01)	05 (0.01)
STRIP	06 (11.28)	14 (267.39)	20 (37.64)
PITPv2.0	10 (492.88)	20 (0.01)	20 (4.23)
PITPv2.1	9 (48.77) [10 takes 625.24]	20 (0.02)	20 (9.14)

Table 3. Refutable ILTP v1.1.2 formulas solved by classes

	SYJ207+1 refutable	SYJ208+1 refutable	SYJ209+1 refutable	SYJ211+1 refutable	SYJ212+1 refutable
ft Prolog	07 (358.05)	08 (65.41)	10 (543.09)	04 (66.62)	20 (0.01)
ft C	07 (51.13)	17 (81.41)	10 (96.99)	04 (17.25)	20 (0.01)
LJT	03 (2.64)	08 (0.18)	10 (461.27)	08 (546.46)	07 (204.98)
STRIP	04 (9.3)	06 (0.24)	10 (132.55)	09 (97.63)	20 (36.79)
PITPv2.0	05 (103.37)	20 (1.58)	10 (221.10)	20 (214.72)	16 (355.11)
PITPv2.1	06 (84.96)	20 (2.15)	10 (223.99)	20 (248.54)	20 (11.52)

Table 4. PITP on 5000 random formulas containing 100 variables

	0-1s	1-10s	10-100s	100-600s	> 600
PITPv2.0	4883(97.7%	41(0.8%)	15(0.3%)	9(0.2%)	52(1.0%)
PITPv2.1	4919(98.4%	12(0.2%)	10(0.2%)	12(0.2%)	47(0.9%)

References

1. Avellone, A., Ferrari, M., Fiorentini, C., Fiorino, G., Moscato, U.: Esbc: an application for computing stabilization bounds. ENTCS 153(1), 23–33 (2006)
2. Avellone, A., Fiorino, G., Moscato, U.: A parallel implementation of a decision procedure for propositional intuitionistic logic. In: Mayer, M.C., Pirri, F. (eds.), TABLEAUX, POSITION PAPERS AND TUTORIALS (2003)
3. Avellone, A., Fiorino, G., Moscato, U.: A new O(n lg n)-space decision procedure for propositional intuitionistic logic. In: Baaz, A. V. M., Makowsky, J. (eds.), volume VIII of Kurt Gödel Society, Collegium Logicum, pp. 17–33 (2004)
4. Avellone, A., Fiorino, G., Moscato, U.: A Tableau Decision Procedure for Propositional Intuitionistic Logic. In Proceedings of the 6th International Workshop on the Implementation of Logic, of CEUR Workshop Proceedings 212, 64–79 (2006)
5. Avellone, A., Fiorentini, C., Fiorino, G., Moscato, U.: A space efficient implementation of a tableau calculus for a logic with a constructive negation. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 488–502. Springer, Heidelberg (2004)
6. Avellone, A., Fiorino, G., Moscato, U.: Optimization techniques for propositional intuitionistic logic and their implementation. submitted to JAL, (2007) At `http://www.dimequant.unimib.it/PITP/index.html`
7. Constable, R.: Implementing Mathematics with the Nuprl Proof Development System. Prentice–Hall, Englewood Cliffs, New Jersey (1986)
8. Egly, U., Schmitt, S.: On intuitionistic proof transformations, their complexity, and application to constructive program synthesis. Fundam. Inform. 39(1-2), 59–83 (1999)
9. Fiorino, G.: Decision procedures for propositional intermediate logics. PhD thesis, Dipartimento di Scienze dell'Informazione, Universita' degli Studi di Milano, Italy (2001)
10. Hähnle, R.: Tableaux and related methods. Handbook of Automated Reasoning, pp. 100–178. Elsevier and MIT Press (2001)
11. Horrocks, I.: Optimising Tableaux Decision Procedures for Description Logics. PhD thesis, University of Manchester (1997)
12. Larchey-Wendling, D., Méry, D., Galmiche, D.: Strip: Structural sharing for efficient proof-search. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 696–700. Springer, Heidelberg (2001)
13. Massacci, F.: Simplification: A general constraint propagation technique for propositional and modal tableaux. In: de Swart, H. (ed.) TABLEAUX 1998. LNCS (LNAI), vol. 1397, pp. 217–232. Springer, Heidelberg (1998)
14. Mendler, M.: Timing analysis of combinational circuits in intuitionistic propositional logic. Formal Methods in System Design 17(1), 5–37 (2000)
15. Raths, T., Otten, J., Kreitz, C.: The ILTP problem library for intuitionistic logic. release v1.1. To appear in JAR (2006)
16. Sahlin, D., Franzén, T., Haridi, S.: An intuitionistic predicate logic theorem prover. J. Log. Comput. 2(5), 619–656 (1992)

KLMLean 2.0: A Theorem Prover for KLM Logics of Nonmonotonic Reasoning

Laura Giordano[1], Valentina Gliozzi[2], and Gian Luca Pozzato[2]

[1] Dipartimento di Informatica - Università del Piemonte Orientale
"A. Avogadro" - via Bellini 25/G - 15100 Alessandria, Italy
laura@mfn.unipmn.it
[2] Dipartimento di Informatica - Università degli Studi di Torino,
corso Svizzera 185 - 10149 Turin - Italy
{gliozzi,pozzato}@di.unito.it

Abstract. We present KLMLean 2.0, a theorem prover for propositional KLM logics of nonmonotonic reasoning. KLMLean 2.0 implements some analytic tableaux calculi for these logics recently introduced. KLMLean 2.0 is inspired by the "lean" methodology, it is implemented in SICStus Prolog and it also contains a graphical interface written in Java[1].

1 Introduction

In the early 90s Kraus, Lehmann and Magidor (from now on KLM) proposed a formalization of nonmonotonic reasoning that has become a point of reference [1,2]. According to KLM framework, a defeasible knowledge base is represented by a (finite) set of nonmonotonic conditionals of the form $A \mid\sim B$, whose reading is *normally (or typically) the A's are B's*. The operator "$\mid\sim$" is nonmonotonic, in the sense that $A \mid\sim B$ does not imply $A \wedge C \mid\sim B$. For instance, a knowledge base K may contain $\mathit{football\_lover} \mid\sim \mathit{bet}$, $\mathit{football\_player} \mid\sim \mathit{football\_lover}$, $\mathit{football\_player} \mid\sim \neg \mathit{bet}$, whose meaning is that people loving football typically bet on the result of a match, football players typically love football but they typically do not bet (especially on matches they are going to play...). If $\mid\sim$ were interpreted as classical implication, one would get $\mathit{football\_player} \mid\sim \bot$, i.e. typically there are not football players, thereby obtaining a trivial knowledge base. In KLM framework, the set of adopted inference rules defines some fundamental types of inference systems, namely, from the strongest to the weakest: Rational (**R**), Preferential (**P**), Loop-Cumulative (**CL**), and Cumulative (**C**) logic. In all these systems one can infer new assertions without incurring the trivializing conclusions of classical logic: in the above example, in none of the systems can one infer $\mathit{football\_player} \mid\sim \mathit{bet}$. In cumulative logics (both **C** and **CL**) one can infer $\mathit{football\_lover} \wedge \mathit{football\_player} \mid\sim \neg \mathit{bet}$, giving preference

[1] This research has been partially supported by "*Progetto Lagrange - Fondazione CRT*" and by the projects "*MIUR PRIN05: Specification and verification of agent interaction protocols*" and *"GALILEO 2006: Interazione e coordinazione nei sistemi multi-agenti"*.

N. Olivetti (Ed.): TABLEAUX 2007, LNAI 4548, pp. 238–244, 2007.

to more specific information; in Preferential logic **P** one can also infer that $\mathit{football\_lover} \mid\!\sim \neg \mathit{football\_player}$; in the rational case **R**, if one further knows that $\neg(\mathit{football\_lover} \mid\!\sim \mathit{rich})$, that is to say it is not the case that football lovers are typically rich persons, one can also infer that $\mathit{football\_lover} \wedge \neg \mathit{rich} \mid\!\sim \mathit{bet}$.

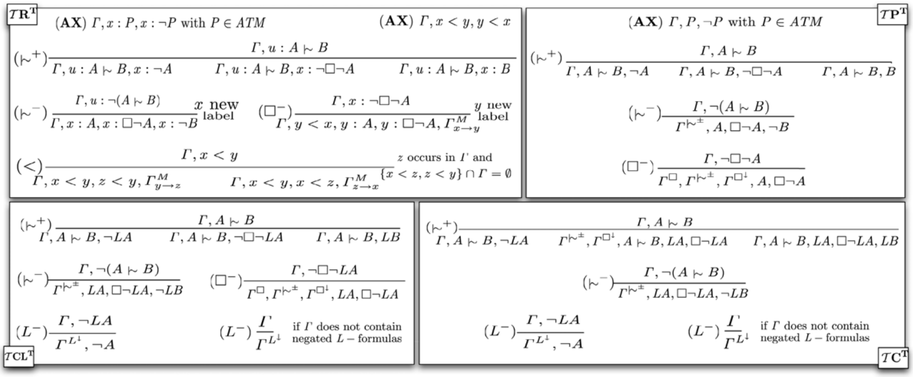
In [3,4,5] analytic tableaux procedures $\mathcal{T}\mathbf{S}^{\mathbf{T}}$ for propositional KLM logics are introduced. In this work we describe an implementation of $\mathcal{T}\mathbf{S}^{\mathbf{T}}$ in SICStus Prolog called **KLMLean 2.0**: it is inspired by the "lean" methodology [6], and it also contains a graphical interface written in Java. For the rational logic **R**, KLMLean 2.0 implements the *labelled* calculus $\mathcal{T}\mathbf{R}^{\mathbf{T}}$ introduced in [5], and offers two different versions: 1. a simple version, where Prolog *constants* are used to represent $\mathcal{T}\mathbf{R}^{\mathbf{T}}$'s labels; 2. a more efficient one, where labels are represented by Prolog *variables*, inspired by the free-variable tableau presented in [7]. To the best of our knowledge, KLMLean 2.0 is the first theorem prover for KLM logics.

2 KLM Logics and Their Tableau Calculi

We consider a propositional language $\mathcal{L}$ defined from a set of propositional variables ATM, the boolean connectives and the conditional operator $\mid\!\sim$. We use $A, B, C, ...$ to denote propositional formulas, whereas $F, G, ...$ are used to denote all formulas (including conditionals). The formulas of $\mathcal{L}$ are defined as follows: if A is a propositional formula, $A \in \mathcal{L}$; if A and B are propositional formulas, $A \mid\!\sim B \in \mathcal{L}$; if F is a boolean combination of formulas of $\mathcal{L}$, $F \in \mathcal{L}$.

In general, the semantics of KLM logics is defined by considering possible world (or possible states) structures with a *preference relation* $w < w'$ among worlds (or states), whose meaning is that w is preferred to w'. $A \mid\!\sim B$ holds in a model $\mathcal{M}$ if B holds in all *minimal worlds* (*states*) where A holds. This definition makes sense provided minimal worlds for A exist whenever there are A-worlds (A-states): this is ensured by the *smoothness condition* defined below. We recall the semantics of KLM logics [1,2] from the strongest **R** to the weakest **C**. A *rational* model is a triple $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$, where $\mathcal{W}$ is a non-empty set of items called worlds, $<$ is an irreflexive, transitive and *modular*[2] relation on $\mathcal{W}$, and V is a valuation function $V : \mathcal{W} \longmapsto 2^{ATM}$. The truth conditions for a formula F are as follows: - if F is a boolean combination of formulas, $\mathcal{M}, w \models F$ is defined as for propositional logic; - let A be a propositional formula; we define $Min_<(A) = \{w \in \mathcal{W} \mid \mathcal{M}, w \models A \text{ and } \forall w',\ w' < w \text{ implies } \mathcal{M}, w' \not\models A\}$; - $\mathcal{M}, w \models A \mid\!\sim B$ if for all w', if $w' \in Min_<(A)$ then $\mathcal{M}, w' \models B$. We also define the *smoothness condition* on the preference relation: if $\mathcal{M}, w \models A$, then $w \in Min_<(A)$ or $\exists w' \in Min_<(A)$ s.t. $w' < w$. Validity and satisfiability of a formula are defined as usual. A *preferential* model is defined as the rational model, with the only difference that the preference relation $<$ is no longer assumed to be modular. Models for (loop-)cumulative logics also comprise states. A *(loop-)cumulative* model is a tuple $\mathcal{M} = \langle S, \mathcal{W}, l, <, V \rangle$, where S is a set of states and $l : S \mapsto 2^{\mathcal{W}}$ is a function that labels every state with a nonempty set of worlds; $<$ is defined on S, it satisfies the smoothness condition and it is irreflexive and transitive in **CL**,

[2] A relation $<$ is modular if, for each u, v, w, if $u < v$, then either $w < v$ or $u < w$.



$\mathcal{T}\mathbf{R}^{\mathbf{T}}$

$(\mathbf{AX})\ \Gamma, x : P, x : \neg P$ with $P \in ATM$ $\qquad (\mathbf{AX})\ \Gamma, x < y, y < x$

$$(\mid\!\sim^{+})\ \frac{\Gamma, u : A \mid\!\sim B}{\Gamma, u : A \mid\!\sim B, x : \neg A \qquad \Gamma, u : A \mid\!\sim B, x : \neg\Box\neg A \qquad \Gamma, u : A \mid\!\sim B, x : B}$$

$$(\mid\!\sim^{-})\ \frac{\Gamma, u : \neg(A \mid\!\sim B)}{\Gamma, x : A, x : \Box\neg A, x : \neg B}\ x \text{ new label} \qquad (\Box^{-})\ \frac{\Gamma, x : \neg\Box\neg A}{\Gamma, y < x, y : A, y : \Box\neg A, \Gamma^{M}_{x\to y}}\ y \text{ new label}$$

$$(<)\ \frac{\Gamma, x < y}{\Gamma, x < y, z < y, \Gamma^{M}_{y\to z} \qquad \Gamma, x < y, x < z, \Gamma^{M}_{z\to x}}\ \begin{array}{l} z \text{ occurs in } \Gamma \text{ and} \\ \{x < z, z < y\} \cap \Gamma = \emptyset \end{array}$$

$\mathcal{T}\mathbf{P}^{\mathbf{T}}$

$(\mathbf{AX})\ \Gamma, P, \neg P$ with $P \in ATM$

$$(\mid\!\sim^{+})\ \frac{\Gamma, A \mid\!\sim B}{\Gamma, A \mid\!\sim B, \neg A \qquad \Gamma, A \mid\!\sim B, \neg\Box\neg A \qquad \Gamma, A \mid\!\sim B, B}$$

$$(\mid\!\sim^{-})\ \frac{\Gamma, \neg(A \mid\!\sim B)}{\Gamma^{\mid\sim^{\pm}}, A, \Box\neg A, \neg B}$$

$$(\Box^{-})\ \frac{\Gamma, \neg\Box\neg A}{\Gamma^{\Box}, \Gamma^{\mid\sim^{\pm}}, \Gamma^{\Box^{\downarrow}}, A, \Box\neg A}$$

$\mathcal{T}\mathbf{CL}^{\mathbf{T}}$

$$(\mid\!\sim^{+})\ \frac{\Gamma, A \mid\!\sim B}{\Gamma, A \mid\!\sim B, \neg LA \qquad \Gamma, A \mid\!\sim B, \neg\Box\neg LA \qquad \Gamma, A \mid\!\sim B, LB}$$

$$(\mid\!\sim^{-})\ \frac{\Gamma, \neg(A \mid\!\sim B)}{\Gamma^{\mid\sim^{\pm}}, LA, \Box\neg LA, \neg LB} \qquad (\Box^{-})\ \frac{\Gamma, \neg\Box\neg LA}{\Gamma^{\Box}, \Gamma^{\mid\sim^{\pm}}, \Gamma^{\Box^{\downarrow}}, LA, \Box\neg LA}$$

$$(L^{-})\ \frac{\Gamma, \neg LA}{\Gamma^{L^{\downarrow}}, \neg A} \qquad (L^{-})\ \frac{\Gamma}{\Gamma^{L^{\downarrow}}}\ \begin{array}{l} \text{if } \Gamma \text{ does not contain} \\ \text{negated } L-\text{formulas} \end{array}$$

$\mathcal{T}\mathbf{C}^{\mathbf{T}}$

$$(\mid\!\sim^{+})\ \frac{\Gamma, A \mid\!\sim B}{\Gamma, A \mid\!\sim B, \neg LA \qquad \Gamma^{\mid\sim^{\pm}}, \Gamma^{\Box^{\downarrow}}, A \mid\!\sim B, LA, \Box\neg LA \qquad \Gamma, A \mid\!\sim B, LA, \Box\neg LA, LB}$$

$$(\mid\!\sim^{-})\ \frac{\Gamma, \neg(A \mid\!\sim B)}{\Gamma^{\mid\sim^{\pm}}, LA, \Box\neg LA, \neg LB}$$

$$(L^{-})\ \frac{\Gamma, \neg LA}{\Gamma^{L^{\downarrow}}, \neg A} \qquad (L^{-})\ \frac{\Gamma}{\Gamma^{L^{\downarrow}}}\ \begin{array}{l} \text{if } \Gamma \text{ does not contain} \\ \text{negated } L-\text{formulas} \end{array}$$

Fig. 1. Tableau systems $\mathcal{T}\mathbf{S}^{\mathbf{T}}$. To save space, we omit the standard rules for boolean connectives. For $\mathcal{T}\mathbf{CL}^{\mathbf{T}}$ and $\mathcal{T}\mathbf{C}^{\mathbf{T}}$ the axiom $(\mathbf{AX})$ is as in $\mathcal{T}\mathbf{P}^{\mathbf{T}}$.

whereas it is only irreflexive in **C**. A propositional formula holds in a state s if it holds in *all* the worlds $w \in l(s)$; a conditional $A \mid\!\sim B$ holds in a model if B holds in all minimal states where A holds.

In Figure 1 we present the tableaux calculi $\mathcal{T}\mathbf{S}^{\mathbf{T}}$ for KLM logics, where S stands for $\{\mathbf{R}, \mathbf{P}, \mathbf{CL}, \mathbf{C}\}$. The basic idea is simply to interpret the preference relation as an accessibility relation. The calculi for **R** and **P** implement a sort of run-time translation into (extensions of) Gödel-Löb modal logic of provability G. This is motivated by the fact that we assume the smoothness condition, which ensures that minimal A-worlds exist whenever there are A-worlds, by preventing infinitely descending chains of worlds. This condition therefore corresponds to the finite-chain condition on the accessibility relation (as in modal logic G). This approach is extended to the cases of **CL** and **C** by using a second modality L which takes care of states. The rules of the calculi manipulate sets of formulas Γ. We write Γ, F as a shorthand for $\Gamma \cup \{F\}$. Moreover, given Γ we define the following sets: $\Gamma^{\Box} = \{\Box\neg A \mid \Box\neg A \in \Gamma\}$; $\Gamma^{\Box^{\downarrow}} = \{\neg A \mid \Box\neg A \in \Gamma\}$; $\Gamma^{\mid\sim^{\pm}} = \{A \mid\!\sim B \mid A \mid\!\sim B \in \Gamma\} \cup \{\neg(A \mid\!\sim B) \mid \neg(A \mid\!\sim B) \in \Gamma\}$; $\Gamma^{L^{\downarrow}} = \{A \mid LA \in \Gamma\}$. As mentioned, the calculus for rational logic **R** makes use of *labelled* formulas, where the labels are drawn from a denumerable set $\mathcal{A}$; there are two kinds of formulas: 1. *world formulas*, denoted by $x : F$, where $x \in \mathcal{A}$ and $F \in \mathcal{L}$; 2. *relation formulas*, denoted by $x < y$, where $x, y \in \mathcal{A}$, representing the preference relation. We define $\Gamma^{M}_{x\to y} = \{y : \neg A, y : \Box\neg A \mid x : \Box\neg A \in \Gamma\}$.

The calculi $\mathcal{T}\mathbf{S}^{\mathbf{T}}$ are sound and complete wrt the semantics, i.e. given a set of formulas Γ of $\mathcal{L}$, it is unsatisfiable if and only if there is a closed tableau in $\mathcal{T}\mathbf{S}^{\mathbf{T}}$ having Γ as a root [3,4,5]. The calculi $\mathcal{T}\mathbf{S}^{\mathbf{T}}$ do not guarantee termination. In order to ensure termination, we have to control the application of the $(\mid\!\sim^{+})$ rule, which can otherwise be applied without any control since it copies its principal formula $A \mid\!\sim B$ in all its conclusions. In [3,4,5], it is shown that it is useless to apply $(\mid\!\sim^{+})$ *more than once in the same world*, therefore the calculi $\mathcal{T}\mathbf{S}^{\mathbf{T}}$ keep track of positive conditionals already considered in a world by moving them in an additional set Σ in the conclusions of $(\mid\!\sim^+)$, and restrict the application of this rule to unused conditionals only. The dynamic rules $(\mid\!\sim^-)$ and $(\Box^-)$, whose conclusions represent a *different* world wrt the corresponding premise, reintroduce formulas from Σ in order to allow further applications of $(\mid\!\sim^+)$ in the other worlds. This machinery is standard. Concerning the labelled calculus $\mathcal{T}\mathbf{R}^{\mathbf{T}}$, the same mechanism is applied by equipping each positive conditional with the list L of worlds-labels in which $(\mid\!\sim^+)$ has already been applied, and restricting its application by using worlds not belonging to L. In [3,4,5] it is shown that no other machinery is needed to ensure termination, except for $\mathcal{T}\mathbf{C}^{\mathbf{T}}$, which needs a further standard loop-checking machinery.

3 Design of KLMLean 2.0

We describe an implementation of $\mathcal{T}\mathbf{S}^{\mathbf{T}}$ calculi in SICStus Prolog. The program, called KLMLean 2.0, is inspired by the "lean" methodology [6] (even if it does not fit its style in a rigorous manner): the Prolog program consists in a set of clauses, each one representing a tableau rule or axiom; the proof search is provided for free by the mere depth-first search mechanism of Prolog, without any additional ad hoc mechanism. KLMLean 2.0 can be downloaded at `http://www.di.unito.it/∼pozzato/klmlean 2.0`.

Let us first describe the implementation of non-labelled calculi for **P**, **CL**, and **C**. We represent each node of a proof tree (i.e. set of formulas) by a Prolog list. The tableaux calculi are implemented by the predicate

prove(Gamma,Sigma,Tree).

which succeeds if and only if the set of formulas Γ, represented by the list `Gamma`, is unsatisfiable. `Sigma` is the list representing the set Σ of *used conditionals*, and it is used in order to control the application of the $(\mid\!\sim^+)$ rule, as described in the previous section. When `prove` succeeds, `Tree` contains a representation of a closed tableau. For instance, to prove that $A \mid\!\sim B \wedge C, \neg(A \mid\!\sim C)$ is unsatisfiable in **P**, one queries KLMLean 2.0 with the following goal: `prove([a => (b and c), neg (a => c)],[ ],Tree).` The string "`=>`" is used to represent the conditional operator $\mid\!\sim$, "`and`" is used to denote $\wedge$, and so on. Each clause of `prove` implements one axiom or rule of the tableaux calculi; for example, the clauses implementing (**AX**) and $(\mid\!\sim^-)$ are as follows:

```
prove(Gamma,_,tree(...)):-member(F,Gamma),member(neg F,Gamma),!.
prove(Gamma,Sigma,tree(...)):-select(neg (A => B),Gamma,NewGamma),
   conditionals(NewGamma,Cond),append(Cond,Sigma,DefGamma),
   prove([neg B|[box neg A|[A|DefGamma]]],[],...).
```

The clause for **(AX)** is applied when a formula F and its negation $\neg F$ belong to Γ. Notice that F is a formula of the language $\mathcal{L}$, even complex; KLMLean 2.0 extends **(AX)** to a generic formula F in order to increase its performances, without losing the soundness of the calculi. The clause for $(\mid\!\sim^-)$ is applied when a formula $\neg(A \mid\!\sim B)$ belongs to Γ. The predicate `select` removes $\neg(A \mid\!\sim B)$ from

Gamma, then the auxiliary predicate conditionals is invoked to compute the set $\Gamma^{\mid\!\sim^{\pm}}$ on the resulting list NewGamma; finally, the predicate prove is recursively invoked on the only conclusion of the rule. Notice that, since ($\mid\!\sim^{-}$) is a dynamic rule, the conditionals belonging to Σ move to Γ in the conclusion (execution of append), in order to allow further applications of ($\mid\!\sim^{+}$). To search for a derivation of a set of formulas Γ, KLMLean 2.0 proceeds as follows: first of all, if Γ is an instance of (**AX**), the goal will succeed immediately by using the clauses for the axioms. If it is not, then the first applicable rule will be chosen, e.g. if Gamma contains a formula neg(neg F), then the clause for ($\neg$) rule will be used, invoking prove on its unique conclusion. KLMLean 2.0 proceeds in a similar way for the other rules. The ordering of the clauses is such that the boolean rules are applied before the other ones. In the case of cumulative logic **C**, KLMLean 2.0 implements a loop-checking machinery by equipping the prove predicate with an additional argument, called Analyzed, representing the list of sets of formulas already considered in the current branch. Clauses implementing $\mathcal{T}\mathbf{C}^{\mathbf{T}}$ are invoked only if the current set of formulas has not yet been considered, i.e. if it does not belong to Analyzed.

The theorem prover for rational logic **R** implements *labelled* tableau calculi $\mathcal{T}\mathbf{R}^{\mathbf{T}}$. It makes use of Prolog constants to represent labels: world formulas $x : A$ are represented by a Prolog list [x,a], and relation formulas $x < y$ are represented by a list [x,<,y]. As for the other systems, each clause of the predicate prove implements a tableau rule or axiom. As an example, here is the clause implementing the rule ($<$), capturing the modularity of the preference relation:

```
prove(Gamma,Labels,Cond,tree(...)):-
  member([X,<,Y],Gamma),member(Z,Labels),X\=Z, Y\=Z,
  \+member([X,<,Z],Gamma),\+member([Z,<,Y],Gamma),!,
  gammaM(Gamma,Y,Z,ResLeft),gammaM(Gamma,Z,X,ResRight),
  append(ResLeft,Gamma,LeftConcl),append(ResRight,Gamma,RightConcl),
  prove([[Z,<,Y]|LeftConcl],Labels,Cond,...),!,
  prove([[X,<,Z]|RightConcl],Labels,Cond,...).
```

The predicate gammaM(Gamma,X,Y,...) computes the set $\Gamma^{M}_{x \rightarrow y}$ defined in the previous section. In this system, Cond is used in order to control the application of the ($\mid\!\sim^{+}$) rule: it is a list whose elements have the form [x,a => b], representing that ($\mid\!\sim^{+}$) has been applied to $A \mid\!\sim B$ in the current branch by using the label x. In order to increase its performances, KLMLean for **R** adopts a heuristic approach (not very "lean") to implement the crucial ($\mid\!\sim^{+}$) rule: the predicate prove *chooses the "best" positive conditional* to which apply the rule, and the *"best" label to use.* Roughly speaking, an application of ($\mid\!\sim^{+}$) is considered to be better than another one if it leads to an immediate closure of more conclusions. Even if ($\mid\!\sim^{+}$) is invertible, choosing the right label in the application of ($\mid\!\sim^{+}$) is highly critical for the performances of the thorem prover. To postpone this choice as much as possible, for the logic **R** we have defined a more efficient version of the prover, inspired by the free-variable tableaux introduced in [7]. It makes use of *Prolog variables* to represent all the labels that can be used in a single application of the $(\mid\!\sim^{+})$ rule. This version represents labels by integers starting from 1; by using integers we can easily express constraints on the range of the variable-labels. To this regard, the library `clpfd` is used to manage free-variables domains. In order to prove $\Gamma, u : A \mid\!\sim B$, KLMLean 2.0 will call `prove` on the following conclusions: $\Gamma, u : A \mid\!\sim B, Y : \neg A$; $\Gamma, u : A \mid\!\sim B, Y : \neg\Box\neg A$; $\Gamma, u : A \mid\!\sim B, Y : B$, where Y is a Prolog variable. Y will then be instantiated by Prolog's pattern matching to close a branch with an axiom. Predicate `Y in 1..Max` is used to define the domain of Y, where `Max` is the maximal integer occurring in the branch (i.e. the last label introduced). The list `Cond` here contains elements of the form `[a => b,Used]`, where `Used` is the list of free variables already introduced to apply $(\mid\!\sim^{+})$ in the current branch. In order to ensure termination, the clause implementing $(\mid\!\sim^{+})$ is applied *only if* |`Used`|< `Max`; the predicate `all_different([Y|Used])` is then invoked to ensure that all variables used to apply $(\mid\!\sim^{+})$ on the same conditional will assume different values. On the unsatisfiable set $\neg(A \vee D \mid\!\sim F \vee \neg C \vee \neg B \vee A), (A \wedge B) \vee (C \wedge D) \mid\!\sim E \wedge F, \neg(P \mid\!\sim E \wedge F), \neg((A \wedge B) \vee (C \wedge D) \mid\!\sim G)$, the free-variables version succeeds in less than 2 ms, whereas the "standard" version requires 1.9 s. The performances of KLMLean 2.0 are promising. We have tested the implementation for **R** over 300 sets of formulas: it terminates its computation in 236 cases in less than 2.5 s (204 in less than 100 ms). The results for **R** and for the other KLM logics are reported in Table 1:

Table 1. Some statistics for KLMLean 2.0

KLM logic	1 ms	10 ms	100 ms	1 s	2.5 s
R (improved version)	176	178	204	223	236
P	166	164	185	206	211
CL	119	118	136	150	159
C	76	77	92	110	123

In future research we intend to increase the performances of KLMLean 2.0 by experimenting standard refinements and heuristics.

References

1. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44(1-2), 167–207 (1990)
2. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence 55(1), 1–60 (1992)
3. Pozzato, G.L.: Proof Methods for Conditional and Preferential Logics. PhD thesis (2007)
4. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Analytic Tableaux for KLM Preferential and Cumulative Logics. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 666–681. Springer, Heidelberg (2005)

5. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Analytic Tableaux Calculi for KLM Rational Logic R. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 190–202. Springer, Heidelberg (2006)
6. Beckert, B., Posegga, J.: leantap: Lean tableau-based deduction. Journal of Automated Reasoning 15(3), 339–358 (1995)
7. Beckert, B., Goré, R.: Free variable tableaux for propositional modal logics. In: Galmiche, D. (ed.) TABLEAUX 1997. LNCS, vol. 1227, pp. 91–106. Springer, Heidelberg (1997)

Author Index